

UNIVERSIDAD NACIONAL DEL CENTRO DEL PERÚ

INGENIERÍA DE SISTEMAS



DESARROLLO DE APLICACIONES WEB

TEMA: CSS AVANZADO

DOCENTE: SUASNABAR TERREL JAIME

ALUMNOS:

- CIPRIANO SILVA DEGGO
- HUAMANI ALVAREZ DEIVI
- MAGRO COLQUI LUIS
- PINARES TAYPE ALVARO
- ROJAS GARCIA DANNY
- SURICHAQUI BARRA JOAO

Huancayo -2025ÍNDICE

1. Variables CSS y Custom Properties:	6
2. Funciones CSS (calc(), clamp(), min(), max())	7
2.1. calc():	7
2.2. clamp():	7
2.3. min():	8
2.4. max():	8
3. Grid Layout avanzado (Áreas de grid, auto-fit, auto-fill)	9
3.1. Áreas de Grid	9
3.2. Auto-fit	9
3.3. Auto-fill	9
4. Flexbox avanzado (Alineaciones, distribución dinámica)	10
4.1. Alineación Avanzada	10
4.1.1. Alineación en el Eje Principal (propiedades en el contenedor flex)	10
4.1.2. Alineación en el Eje Transversal (propiedades en el contenedor flex)	10
4.1.3. Alineación Individual (propiedad en los elementos flex)	11
4.2. Distribución Dinámica	11
5. Diseño responsivo (Media Queries y Mobile-first)	13
5.1. Fundamentos del Diseño Responsivo	13
5.2. Media Queries en CSS	13
5.3. Enfoque Mobile-First	14
5.4. Beneficios del Diseño Responsivo	15
6. Scroll Snap y Técnicas de Scroll	15
6.1. Control del Scroll en la Web	15
6.2. Scroll Snap: Alineación Precisa	16
6.3. Scroll Suave (Smooth Scrolling)	16
6.4. Efectos Visuales Sincronizados	17
6.5. Consideraciones de Accesibilidad y Rendimiento	17
7. Clipping y Masking con CSS	17
7.1. Propiedades de Clipping	18
7.2. Propiedades de Masking	19
7.2.1. Propiedad mask y subpropiedades	19
7.2.2. Máscaras con -webkit-	19
7.2.3. Diferencias con clipping	20
8. Propiedades y Contenedores CSS (Container Queries)	20
8.1. Fundamentos de Container Queries	21
8.1.1. Propiedades de contención	21
8.1.2. Regla @container	21
8.2. Unidades de Container Queries	22
9. Filtros y Efectos Visuales avanzados	22
9.1. blur()	23

9.2. brightness()	24
9.3. contrast()	24
9.4. grayscale()	25
9.5. hue-rotate ()	26
9.6. invert()	27
9.7. opacity()	28
9.8. Filtros Múltiples	28
9.9. Ventajas del Uso de Filtros en Desarrollo Web	29
10. Técnicas de Performance y Optimización en CSS	30
10.1. Minificar el CSS	30
10.3. Usar el cargado condicional de CSS	31
10.4. Uso de variable CSS para consistencia	31
10.5. Combinar archivos CSS	32
10.6. Uso eficiente de los media queries	32
10.7. Prefetching y prerendering de CSS	33
10.8. Reducción del uso de selectores complejos	33

Introducción

En la actualidad, el desarrollo web se ha convertido en una disciplina esencial dentro del ámbito tecnológico, donde la experiencia del usuario y el diseño visual juegan un rol determinante en el éxito de los productos digitales. En este contexto, las hojas de estilo en cascada, conocidas como CSS (Cascading Style Sheets), constituyen una herramienta fundamental para definir la apariencia y presentación de los sitios web. Mientras que el CSS básico permite aplicar estilos generales como colores, tipografías y márgenes, el uso de CSS avanzado abre un abanico de posibilidades que optimizan la estructura, adaptabilidad y estética de las interfaces. Este trabajo tiene como finalidad abordar el tema de CSS avanzado, analizando sus principales características, herramientas modernas, así como su aplicación práctica en proyectos reales.

El objetivo general de este trabajo es analizar y aplicar técnicas de CSS avanzado en el diseño y desarrollo de interfaces web modernas, optimizando la experiencia del usuario y la eficiencia del código. Para alcanzar esta meta, se plantean los siguientes objetivos específicos: describir las principales características y ventajas del uso de CSS avanzado; aplicar buenas prácticas en la organización y mantenimiento de hojas de estilo; y finalmente, diseñar un prototipo web que implemente herramientas avanzadas de CSS.

Este tema reviste una gran importancia debido a la creciente necesidad de crear sitios web funcionales, atractivos y adaptables a diversos dispositivos. Dominar CSS avanzado no solo permite mejorar la apariencia visual de una página, sino que también aporta a la accesibilidad, el rendimiento y la mantenibilidad del código. La correcta implementación de estas técnicas contribuye directamente a una mejor experiencia del usuario, lo cual es clave en el entorno competitivo del desarrollo web actual. Además, el conocimiento profundo de CSS avanzado fortalece el perfil profesional del desarrollador, permitiéndole enfrentar proyectos de mayor complejidad con mayor eficiencia.

Desarrollo o cuerpo principal

El lenguaje CSS (Cascading Style Sheets) ha evolucionado considerablemente desde sus primeras versiones. Lo que antes era simplemente una manera de aplicar color, tamaño y márgenes a los elementos HTML, ahora permite manejar aspectos avanzados como animaciones, diseño responsivo, e incluso lógica básica con funciones matemáticas y variables.

En esta monografía abordaremos diez de los aspectos más importantes del CSS avanzado:

1. Variables CSS y Custom Properties:

En CSS, las propiedades personalizadas (también conocidas como variables) son entidades definidas por autores de CSS que contienen valores específicos que se pueden volver a utilizar en un documento. Se establecen mediante la notación de propiedades personalizadas (por ejemplo, `--main-color: black;`) y se acceden mediante la función `var()` (por ejemplo, `color: var(--main-color);`).*(Uso De Propiedades Personalizadas (Variables) En CSS - CSS / MDN, 2025)*



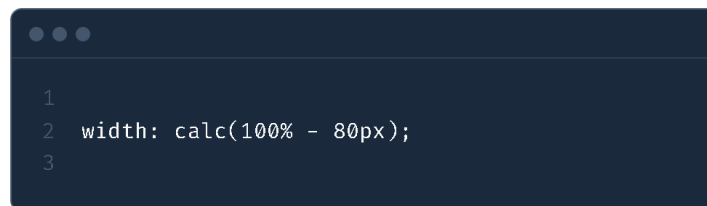
```
1
2 :root {
3   --main-bg-color: #ff0080;
4 }
5 .four, .five {
6   background-color: var(--main-bg-color);
7 }
8
```

`:root` permite definir variables globales en CSS que son accesibles en todo el documento, funcionando como el punto de partida para establecer valores reutilizables. Sin embargo, estas variables pueden ser redefinidas dentro de cualquier selector, lo que permite ajustar su valor localmente según el contexto, aprovechando la herencia y el modelo de cascada que caracteriza a CSS. Esta flexibilidad facilita un estilo más modular, mantenible y dinámico.

2. Funciones CSS (calc(), clamp(), min(), max())

2.1. calc():

Es posible que en algunas ocasiones necesitemos indicar valores pre calculados por el navegador, ya porque sea más cómodo o porque simplemente queremos hacer el código más organizado o fácil de leer y mantener. Por ejemplo, podríamos tener un caso donde tenemos que sumar dos valores que a priori el desarrollador desconoce. Esto puede ocurrir frecuentemente, ya que cuando hacemos una web, esa web será visualizada tanto por usuarios de escritorio con pantallas gigantes, como usuarios de móvil con pantallas pequeñas. (*Calculos Matemáticos En CSS - CSS En Español*, n.d.)

A dark-themed code editor window with three small circles in the top-left corner. It contains three lines of CSS code: line 1 is empty, line 2 is 'width: calc(100% - 80px);', and line 3 is empty.

```
1  
2 width: calc(100% - 80px);  
3
```

2.2. clamp():

Esta necesariamente debe recibir 3 parámetros y en un orden concreto: VALOR_MINIMO, VALOR_IDEAL y VALOR_MAXIMO. La función se resuelve con la fórmula: $\max(\text{VALOR_MINIMO}, \min(\text{VALOR_IDEAL}, \text{VALOR_MAXIMO}))$. Y si le pusiéramos palabras podríamos leerla como: “Escoge el valor mayor entre el mínimo (que funciona como límite por debajo) y el máximo que haya surgido tras resolverse la función min() entre el valor ideal y el máximo (que funcionar como límite por arriba)”. (Cagigao & Mur, 2020)

```
1 font-size: clamp(14px, 3em, 4rem);  
2
```

2.3. min():

La función min() toma uno o más expresiones separadas por coma como sus parámetros, y usa el valor más pequeño de esas expresiones como su valor. Las expresiones pueden ser expresiones matemáticas (usando operadores aritméticos), valores literales, u otras expresiones, tales como attr(), que se evalúan a un tipo de argumento válido (like <length>). Se pueden usar distintas unidades de medida para cada valor en la expresión, si se desea. También puede usar paréntesis para establecer orden de precedencia si lo requiere.(Min() - CSS / MDN, 2025)

```
1 width: min(50vw, 200px);  
2
```

2.4. max():

max() es otra función de CSS que puede aplicarse a cualquier propiedad que soporte alguno de los tipos <length>, <frequency>, <angle>, <time>, <percentage>, <number>, o <integer>. De entre los parámetros separados por comas que recibe la función, escoge el mayor, que será el que se aplique como valor de la propiedad.(Cagigao & Mur, 2020)

```
1 width: max(40em, 98%);  
2
```

3. Grid Layout avanzado (Áreas de grid, auto-fit, auto-fill)

CSS Grid Layout es un sistema de diseño bidimensional que te permite organizar elementos en filas y columnas. Las características avanzadas llevan esta capacidad a un nivel superior:

3.1. Áreas de Grid

Esta propiedad te permite definir un layout visual para tu grid nombrando celdas y luego asignando elementos a esas áreas con la propiedad `grid-area`. Es una forma muy intuitiva de definir la estructura general de tu diseño directamente en la hoja de estilos.

3.2. Auto-fit

Intenta ajustar el tamaño de las columnas (o filas) para que ocupen todo el espacio disponible en el contenedor grid, **después** de haber colocado todos los elementos. Si hay espacio extra, las columnas (o filas) con auto-fit se expandirán para llenarlo. Las columnas vacías colapsarán a 0 ancho.

3.3. Auto-fill

Intenta crear tantas columnas (o filas) como quepan en el contenedor grid, **incluso si están vacías**. Luego, distribuye el espacio entre las columnas (o filas) creadas. Las columnas vacías mantendrán su ancho definido (si lo tienen).


```
1 .contenedor-grid-responsive {
2   display: grid;
3   /* Crea tantas columnas de al menos 150px como quepan */
4   grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
5   gap: 10px;
6 }
7
8 .contenedor-grid-flexible {
9   display: grid;
10  /* Crea tantas columnas de al menos 150px como quepan, incluso si están vacías */
11  grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
12  gap: 10px;
13 }
```

4. Flexbox avanzado (Alineaciones, distribución dinámica)

CSS Flexible Box Layout (Flexbox) es un sistema de diseño unidimensional que facilita la distribución y alineación de elementos dentro de un contenedor. Las técnicas avanzadas de Flexbox permiten crear layouts dinámicos y adaptables con gran control.

4.1. Alineación Avanzada

Flexbox ofrece un conjunto robusto de propiedades para controlar la alineación de los elementos tanto en el eje principal (normalmente horizontal) como en el eje transversal (normalmente vertical).

4.1.1. Alineación en el Eje Principal (propiedades en el contenedor flex)

- **justify-content:** Controla cómo se distribuye el espacio entre y alrededor de los elementos flex a lo largo del eje principal. Valores comunes: flex-start, flex-end, center, space-between, space-around, space-evenly.

4.1.2. Alineación en el Eje Transversal (propiedades en el contenedor flex)

- **align-items:** Controla la alineación de los elementos flex en el eje transversal. Se aplica a todos los elementos flex como grupo. Valores comunes: flex-start, flex-end, center, baseline, stretch.

- **align-content:** Controla la distribución del espacio entre las *líneas* de elementos flex cuando hay varias líneas (es decir, cuando los elementos envuelven). Valores comunes: flex-start, flex-end, center, space-between, space-around, stretch.

4.1.3. Alineación Individual (propiedad en los elementos flex)

- **align-self:** Permite anular la alineación definida por align-items para un elemento flex específico. Los valores son los mismos que para align-items.

```
1  .contenedor-flex {
2    display: flex;
3    justify-content: space-between; /* Espacio igual entre los elementos */
4    align-items: center; /* Centra los elementos verticalmente */
5    height: 200px;
6    background-color: #eee;
7  }
8
9  .item-1 {
10   align-self: flex-start; /* Este elemento se alinea arriba */
11 }
12
13 .item-3 {
14   align-self: flex-end; /* Este elemento se alinea abajo */
15 }
```

4.2. Distribución Dinámica

Flexbox es excelente para crear layouts donde el tamaño y la distribución de los elementos se ajustan dinámicamente según el contenido y el espacio disponible.

- **flex-grow:** Define la capacidad de un elemento flex para crecer y ocupar el espacio extra disponible en el contenedor flex. Un valor mayor indica que el elemento ocupará una proporción mayor del espacio extra.


- **flex-shrink:** Define la capacidad de un elemento flex para encogerse si no hay suficiente espacio en el contenedor flex. Un valor mayor indica que el elemento se encogerá más rápidamente en comparación con otros elementos.
- **flex-basis:** Define el tamaño inicial de un elemento flex antes de que se distribuya el espacio disponible. Puede ser un valor de longitud (px, em, etc.) o auto (el tamaño basado en el contenido).

```
1  .contenedor-flex-dinamico {
2    display: flex;
3    width: 500px;
4    background-color: #f9f9f9;
5  }
6
7  .caja {
8    background-color: #ccc;
9    color: white;
10   padding: 20px;
11   margin: 5px;
12 }
13
14 .caja-1 {
15   flex-grow: 1; /* Crece para ocupar el espacio restante */
16 }
17
18 .caja-2 {
19   flex-grow: 2; /* Crece el doble que la caja 1 */
20 }
21
22 .caja-3 {
23   flex-basis: 100px; /* Tamaño inicial de 100px */
24   flex-shrink: 0; /* No se encoge */
25 }
```

5. Diseño responsivo (Media Queries y Mobile-first)

5.1. Fundamentos del Diseño Responsivo

El diseño responsivo (responsive design) es una estrategia de desarrollo web que busca que un sitio se adapte automáticamente al tamaño, orientación y resolución de la pantalla del dispositivo desde el cual se accede. Esto permite que el mismo sitio web funcione correctamente tanto en teléfonos móviles como en tabletas, laptops o monitores de escritorio, mejorando la experiencia del usuario y reduciendo la necesidad de crear versiones separadas para cada dispositivo [6].

A dark-themed code editor window with three window control buttons (red, yellow, green) in the top-left corner. It contains four lines of CSS code:

```
1  img {  
2    max-width: 100%;  
3    height: auto;  
4  }
```

Nota: Este código asegura que las imágenes escalen de forma proporcional dentro de su contenedor, ajustándose automáticamente al ancho disponible.

5.2. Media Queries en CSS

Las media queries son una de las herramientas clave para implementar diseño responsivo. Permiten aplicar estilos condicionales dependiendo de las características del dispositivo, como el ancho de la pantalla, la orientación (portrait o landscape) o la resolución.

```
1  /* Estilos generales (mobile-first) */
2  body {
3      background-color: white;
4      font-size: 16px;
5  }
6
7  /* Estilos para pantallas medianas (768px en adelante) */
8  @media (min-width: 768px) {
9      body {
10         font-size: 18px;
11     }
12 }
13
14 /* Estilos para pantallas grandes (1200px en adelante) */
15 @media (min-width: 1200px) {
16     body {
17         font-size: 20px;
18     }
19 }
```

Nota: Este enfoque escalonado permite personalizar el diseño a medida que aumenta el tamaño del dispositivo.

5.3. Enfoque Mobile-First

El enfoque mobile-first consiste en diseñar pensando primero en los dispositivos móviles, que tienen más restricciones de espacio y capacidad de procesamiento, y luego expandir hacia pantallas más grandes utilizando media queries. Esta estrategia obliga a priorizar la simplicidad, el contenido esencial y la velocidad de carga [5].

```
1  .container {
2    display: grid;
3    grid-template-columns: 1fr; /* Móvil: una columna */
4  }
5
6  @media (min-width: 768px) {
7    .container {
8      grid-template-columns: 1fr 1fr; /* Tablet: dos columnas */
9    }
10
11  @media (min-width: 1200px) {
12    .container {
13      grid-template-columns: 1fr 1fr 1fr; /* Escritorio: tres columnas */
14  }
```

5.4. Beneficios del Diseño Responsivo

El uso de diseño responsivo trae múltiples beneficios, como:

- Mejora la accesibilidad del contenido.
- Reduce los tiempos de carga en dispositivos móviles.
- Disminuye la tasa de rebote.
- Facilita el mantenimiento del sitio, ya que solo se requiere una versión del código.
- Favorece el posicionamiento en buscadores (SEO), ya que Google prioriza sitios móviles.

6. Scroll Snap y Técnicas de Scroll

6.1. Control del Scroll en la Web

El desplazamiento o scroll es una de las interacciones más frecuentes al navegar en internet. Las técnicas modernas permiten transformar esta acción básica en una experiencia visual más fluida, interactiva y agradable para el usuario. Mediante CSS es posible controlar cómo se comportan los elementos durante el desplazamiento, mejorando la navegabilidad del sitio [4].

6.2. Scroll Snap: Alineación Precisa

Scroll Snap es una funcionalidad de CSS que permite "anclar" automáticamente elementos dentro de un contenedor mientras se hace scroll. Esto es útil para carruseles, secciones por pantalla o listas horizontales.

```
1  .container {  
2    display: flex;  
3    overflow-x: scroll;  
4    scroll-snap-type: x mandatory;  
5  }  
6  
7  .item {  
8    flex: 0 0 100%;  
9    scroll-snap-align: start;  
10 }
```

Nota: En este ejemplo, al deslizarse horizontalmente, cada `.item` se alinea automáticamente al inicio del contenedor.

6.3. Scroll Suave (Smooth Scrolling)

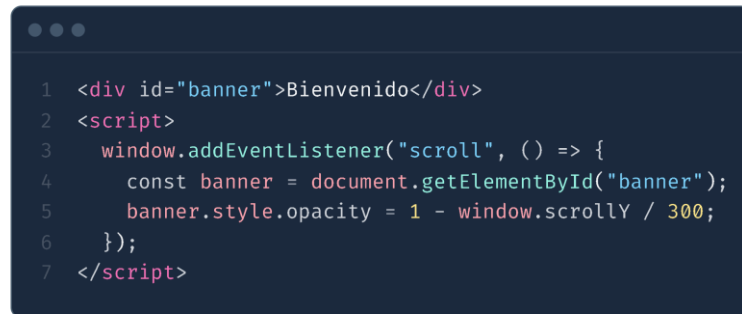
CSS también permite suavizar el comportamiento del scroll para anclas internas, generando transiciones más agradables al navegar entre secciones de la misma página.

```
1  html {  
2    scroll-behavior: smooth;  
3  }
```

Nota: Este código aplica desplazamientos animados al hacer clic en enlaces internos (por ejemplo, ``).

6.4. Efectos Visuales Sincronizados

Mediante la combinación de CSS con JavaScript, es posible aplicar efectos que reaccionan al desplazamiento, como cambios de opacidad, transformaciones, o efectos de paralaje (donde elementos en el fondo se mueven más lento que los del frente).



```
1 <div id="banner">Bienvenido</div>
2 <script>
3   window.addEventListener("scroll", () => {
4     const banner = document.getElementById("banner");
5     banner.style.opacity = 1 - window.scrollY / 300;
6   });
7 </script>
```

Nota: Este fragmento reduce la opacidad del elemento #banner conforme el usuario se desplaza hacia abajo.

6.5. Consideraciones de Accesibilidad y Rendimiento

Si bien estas técnicas mejoran la experiencia visual, es fundamental aplicarlas con moderación. El exceso de animaciones o efectos puede afectar la accesibilidad (por ejemplo, para personas con sensibilidad a movimientos rápidos) y el rendimiento, especialmente en dispositivos móviles con recursos limitados [3].

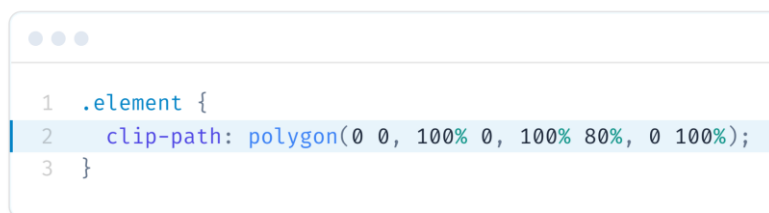
7. Clipping y Masking con CSS

Clipping y masking son técnicas de CSS que permiten manipular la visibilidad de elementos HTML. El clipping recorta un elemento para mostrar solo una porción definida, utilizando formas geométricas o SVG, mediante la propiedad clip-path. Por otro lado, el masking aplica máscaras que controlan la opacidad y visibilidad de un elemento, usando imágenes, gradientes o SVG a través de la propiedad mask. Estas

técnicas son esenciales para crear diseños visuales complejos, como formas personalizadas o efectos de transición.

7.1. Propiedades de Clipping

La propiedad `clip-path` define una región visible de un elemento, recortando el resto. Soporta valores como `circle()`, `ellipse()`, `polygon()`, y referencias a SVG. Por ejemplo:

A code editor window with a light blue header and three dots. The code is as follows:

```
1 .element {  
2   clip-path: polygon(0 0, 100% 0, 100% 80%, 0 100%);  
3 }
```

Fig. 1 Aplicación de la propiedad `clip-path`.

Este código recorta un elemento en forma de trapecio. También se pueden usar SVG para formas complejas:

A code editor window with a light blue header and three dots. The code is as follows:

```
1 .element {  
2   clip-path: url(#myClip);  
3 }
```

Fig. 2 Aplicación de la propiedad `clip-path` con un recurso externo.


En 2025, `clip-path` tiene soporte amplio en navegadores modernos (Chrome, Firefox, Safari), pero formas complejas con `polygon()` pueden requerir pruebas

para garantizar consistencia. Algunas funciones avanzadas, como animaciones, pueden necesitar prefijos en navegadores antiguos.

7.2. Propiedades de Masking

7.2.1. Propiedad mask y subpropiedades

La propiedad mask y sus subpropiedades (mask-image, mask-mode, mask-position) permiten aplicar máscaras que controlan la visibilidad y opacidad. Por ejemplo:



```
1 .element {  
2   mask-image: linear-gradient(to right, black, transparent);  
3 }
```

Fig. 3 Aplicación de la propiedad clip-path con un recurso externo.

Este código crea un desvanecimiento horizontal. Las máscaras también pueden usar imágenes PNG o SVG:



```
1 .element {  
2   mask-image: url('mask.svg');  
3   mask-mode: alpha;  
4 }
```

Fig. 4 Aplicación de la propiedad clip-path con un recurso externo.

7.2.2. Máscaras con -webkit-

En navegadores basados en WebKit (Safari, versiones antiguas de Chrome), mask puede requerir el prefijo -webkit-. Por ejemplo:

A code editor window with a light blue header and three dots in the top left corner. The code is as follows:

```
1 .element {  
2   -webkit-mask-image: url('mask.png');  
3   mask-image: url('mask.png');  
4 }
```

Fig. 4 Aplicación de la propiedad clip-path con un recurso externo.

En 2025, el soporte sin prefijos es casi universal, pero el prefijo sigue siendo relevante para compatibilidad retroactiva .

7.2.3. Diferencias con clipping

El clipping elimina partes de un elemento de forma rígida, mientras que el masking permite transiciones suaves mediante opacidad. Por ejemplo, una máscara con gradiente puede desvanecer un elemento, algo imposible con clip-path .

8. Propiedades y Contenedores CSS (Container Queries)

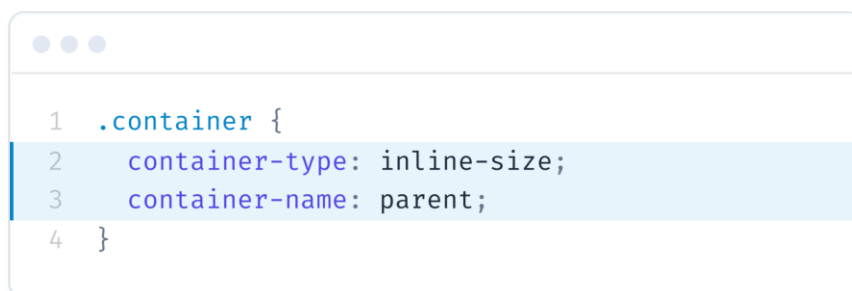
Las container queries (@container) permiten aplicar estilos a elementos según el tamaño de su contenedor, no del viewport, a diferencia de las media queries. Introducidas en CSS3, son clave para el diseño responsivo moderno, ya que hacen los componentes más reutilizables y adaptables [4].

Las container queries facilitan la creación de componentes que se ajustan automáticamente a diferentes contextos (ej., barras laterales, tarjetas). Esto reduce la dependencia de breakpoints fijos y mejora la modularidad [6].

8.1. Fundamentos de Container Queries

8.1.1. Propiedades de contención

La propiedad `container-type` define el tipo de contención (`size`, `inline-size`, `normal`), mientras que `container-name` asigna un nombre al contenedor. Ejemplo:



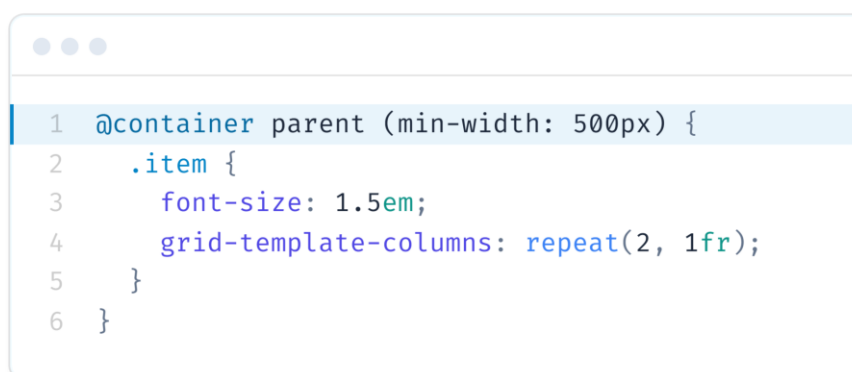
```
1 .container {  
2   container-type: inline-size;  
3   container-name: parent;  
4 }
```

Fig. 4 Aplicación de la propiedad `clip-path` con un recurso externo.

Aquí establecemos un contexto en base al ancho del contenedor.

8.1.2. Regla `@container`

La regla `@container` aplica estilos condicionalmente según el tamaño del contenedor. Ejemplo:



```
1 @container parent (min-width: 500px) {  
2   .item {  
3     font-size: 1.5em;  
4     grid-template-columns: repeat(2, 1fr);  
5   }  
6 }
```

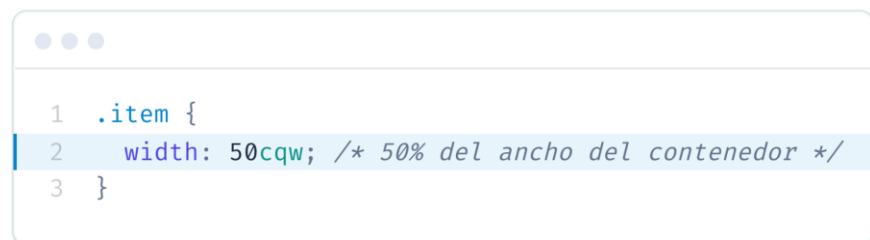
Fig. 4 Aplicación de la propiedad clip-path con un recurso externo.

Este código ajusta el tamaño de fuente y el layout si el contenedor supera los 500px de ancho.

8.2. Unidades de Container Queries

- Unidades relativas

Las unidades como cqw (1% del ancho del contenedor), cqh (1% de la altura), cqi (1% del eje inline) y cqb (1% del eje block) permiten medidas dinámicas. Ejemplo:

A screenshot of a code editor window with a light blue border and three dots in the top-left corner. The editor contains three lines of CSS code. The second line is highlighted with a light blue background. The code is as follows:

```
1 .item {  
2   width: 50cqw; /* 50% del ancho del contenedor */  
3 }
```

Fig. 4 Aplicación de la propiedad clip-path con un recurso externo.

- Comparación con otras unidades

A diferencia de vw o vh (relativas al viewport), las unidades cq* dependen del contenedor, lo que las hace ideales para componentes reutilizables. Comparadas con %, evitan problemas de herencia en elementos anidados [4].

9. Filtros y Efectos Visuales avanzados

En el desarrollo web moderno, los filtros y efectos visuales desempeñan un papel crucial en la mejora de la experiencia del usuario y en la presentación estética de las

interfaces. Gracias a las propiedades avanzadas de CSS, como `filter` y `backdrop-filter`, es posible aplicar transformaciones visuales sofisticadas directamente sobre elementos HTML, sin necesidad de recurrir a herramientas externas de edición gráfica. Estas propiedades permiten implementar efectos como desenfoques, cambios de tonalidad, ajustes de brillo o contraste, entre otros, ofreciendo una estética moderna y funcional, como es el caso del popular efecto glassmorphism.

Principales Filtros CSS

A continuación, se detallan algunos de los filtros más utilizados y sus respectivas funcionalidades:

9.1. `blur()`

El filtro **blur** o desenfoque aplica un desenfoque gaussiano a una imagen. Recibe un parámetro **radio** que representa la cantidad de desenfoque que queremos. A mayor radio, mayor desenfoque. El **radio** solo admite unidades relativas o fijas pero no porcentajes.

```
body {
  margin:0;
  padding:0;
  box-sizing:border-box;
}

.container {
  display:flex;
  justify-content:center;
  align-items:center;
  gap:2em;
}

/* filtros */
figure > .img-filtro-1{
  filter:blur(1px)
}

figure > .img-filtro-2{
  filter:blur(3px)
}

figure > .img-filtro-3{
  filter:blur(5px)
}

figure{
  margin:0;
  padding:10px;
}
```



Mick Jagger



Mick Jagger - blur(1px)



Kurt Cobain



Kurt Cobain - blur(3px)

9.2. brightness()

El filtro de brillo se encarga de oscurecer o aclarar una imagen. Recibe un parámetro que puede ser un porcentaje o un valor escalar.

Si es un porcentaje:

- Un valor menor a 100% oscurece la imagen.
- Un valor mayor a 100% aclara la imagen.

Si es un escalar:

- Un escalar pequeño oscurece la imagen (0 totalmente oscuro).
- Un escalar grande aclara la imagen.

```
body {
  margin:0;
  padding:0;
  box-sizing:border-box;
}


.container {
  display:flex;
  justify-content:center;
  align-items:center;
  gap:2em;
}

/* filtros */
figure > .img-filtro-1{
  filter: brightness(50%);
}

figure > .img-filtro-2{
  filter: brightness(200%);
}

figure > .img-filtro-3{
  filter: brightness(0);
}

figure{
  margin:0;
  padding:0;
}
```



Mick Jagger

Mick Jagger -
brightness(40%)

Kurt Cobain

Kurt Cobain -

9.3. contrast()

Ajusta el contraste de una imagen.

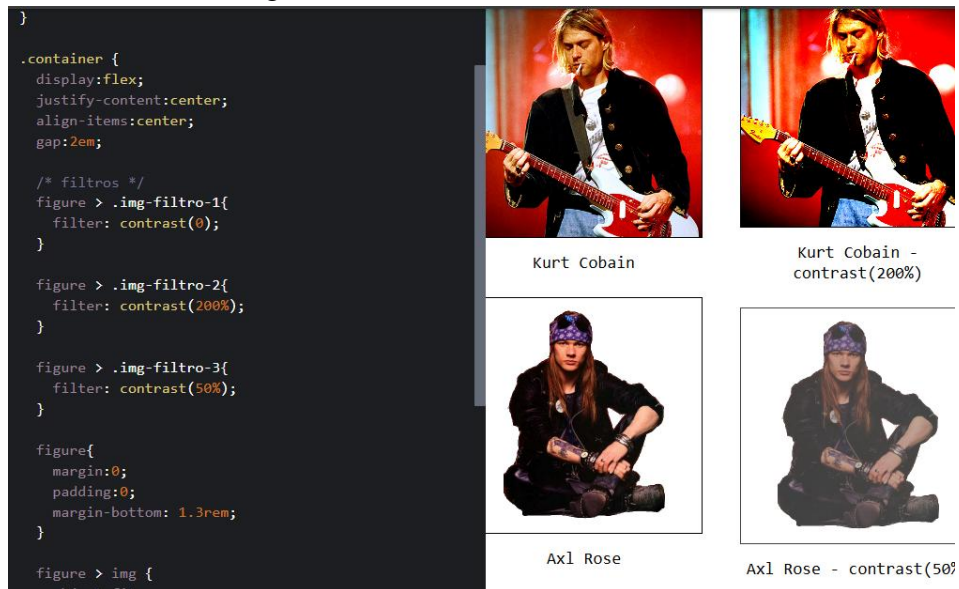
Si es un porcentaje:

- Un valor menor a 100% disminuye el contraste.

- Un valor mayor a 100% aumenta el contraste.

Si es un escalar:

- Un escalar pequeño disminuye el contraste (0 totalmente gris).
- Un escalar grande aumenta el contraste.



9.4. grayscale()

La escala de grises sirve para poner a una imagen en blanco o negro.

Recibe un parámetro que indica cuál será el nivel de la escala de grises.

Si es un porcentaje:

- 0% no afecta en nada a la imagen.
- 100% la convierte totalmente en blanco y negro. > No recibe porcentajes más grandes de 100%

Si es un número es la representación decimal del porcentaje, donde .5 es 50%, .75 es 75%, etc.


```
body {
  margin:0;
  padding:0;
  box-sizing:border-box;
}

.container {
  display:flex;
  justify-content:center;
  align-items:center;
  gap:2em;
}

/* filtros */
figure > .img-filtro-1{
  filter: grayscale(0);
}

figure > .img-filtro-2{
  filter: grayscale(50%);
}

figure > .img-filtro-3{
  filter: grayscale(100%);
}

figure{
  margin:0;
  padding:0;
}
```



Kurt Cobain



Kurt Cobain -
grayscale(50%)



Axl Rose



Axl Rose -

9.5. hue-rotate ()

Permite aplicar un filtro de color recibiendo un ángulo como parámetro.

Si el ángulo:

- Es positivo aumenta el tono de color.
- Es negativo disminuye el tono de color.
- El parámetro puede ser en deg, rad, etc.

```
body {
  margin:0;
  padding:0;
  box-sizing:border-box;
}

.container {
  display:flex;
  justify-content:center;
  align-items:center;
  gap:2em;

  /* filtros */
  figure > .img-filtro-1{
    filter: hue-rotate(90deg);
  }

  figure > .img-filtro-2{
    filter: hue-rotate(-120deg);
  }

  figure > .img-filtro-3{
    filter: hue-rotate(300deg);
  }

  figure{
    margin:0;
    padding:0;
  }
}
```



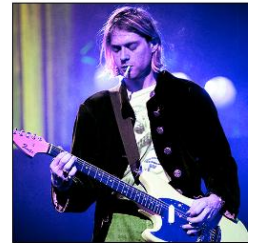
Mick Jagger



Mick Jagger - hue-rotate(90deg)



Kurt Cobain



Kurt Cobain - hue-

9.6. invert()

Invierte los colores de una imagen en base el círculo cromático.

Recibe un porcentaje como parámetro que indica el grado de inversión que tendrá la imagen.

```
body {
  margin:0;
  padding:0;
  box-sizing:border-box;
}

.container {
  display:flex;
  justify-content:center;
  align-items:center;
  gap:2em;

  /* filtros */
  figure > .img-filtro-1{
    filter: invert(100%);
  }

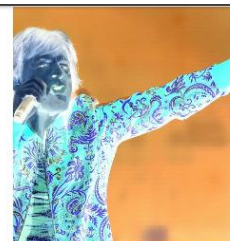
  figure > .img-filtro-2{
    filter: invert(.5); /*50%*/
  }

  figure > .img-filtro-3{
    filter: invert(.3);
  }

  figure{
    margin:0;
    padding:0;
  }
}
```



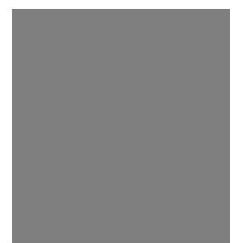
Mick Jagger



Mick Jagger - invert(100%)



Kurt Cobain



Kurt Cobain - invert(50)

9.7. opacity()

Cambia el nivel de transparencia de una imagen. Es muy similar a la propiedad opacity pero la diferencia radica en que cuando usamos filtros es mas optimo por la aceleración de hardware que nos ofrece.

Recibe un porcentaje como parámetro donde a menor opacidad mas difuminada se verá la imagen.

```
margin:0;
padding:0;
box-sizing:border-box;
}

.container {
display:flex;
justify-content:center;
align-items:center;
gap:2em;


/* filtros */
figure > .img-filtro-1{
filter: opacity(.25);
}

figure > .img-filtro-2{
filter: opacity(.5);
}

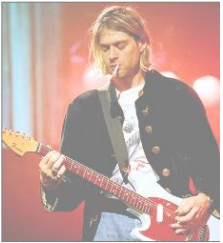
figure > .img-filtro-3{
filter: opacity(0);
}

figure{
margin:0;
padding:0;
margin-bottom: 1.3em;
```


Mick Jagger




opacity(25%)



Kurt Cobain



Kurt Cobain - opacity(50%)



9.8. Filtros Múltiples

Es posible usar múltiples filtros al mismo tiempo, combinándolos para encontrar soluciones más óptimas a problemas concretos.

Un buen ejemplo es para mejorar la accesibilidad de un texto, veamos un ejemplo:



9.9. Ventajas del Uso de Filtros en Desarrollo Web

El uso de filtros en nuestros proyectos son más que recomendados por los siguientes puntos:

- Permite la edición de imágenes sin usar programas como Photoshop o similares.
- Su implementación es muy sencilla.
- Permite uso de pseudo clases como hover y filtros.
- Mejora la accesibilidad.
- Permite escribir sombras más avanzadas con drop-shadow().
- Algunos filtros tienen aceleración por hardware lo que significa que hacen uso de la GPU y no cargan tanto a la CPU del dispositivo brindando una capa extra de optimización.

10. Técnicas de Performance y Optimización en CSS

A medida que los sitios web aumentan en tamaño y complejidad, resulta fundamental aplicar técnicas de optimización al código CSS para garantizar una carga eficiente y una experiencia de usuario fluida. Un CSS mal estructurado o excesivamente extenso puede impactar negativamente en el rendimiento del sitio, por lo que es indispensable adoptar buenas prácticas que reduzcan el peso del archivo, disminuyan las solicitudes al servidor y faciliten el mantenimiento del código.

10.1. Minificar el CSS

La minificación consiste en eliminar espacios, saltos de línea y comentarios innecesarios del código CSS, lo que disminuye su tamaño y acelera la carga de la página.

- npm install cssnano

```
1  /* Antes de minificar */
2  body {
3      margin: 0;
4      padding: 0;
5      font-family: 'Arial', sans-serif;
6  }
7
8  /* Después de minificar */
9  body{margin:0;padding:0;font-family:'Arial',sans-serif;}
```

La minificación elimina espacios en blanco, comentarios y saltos de línea innecesarios para reducir el tamaño del archivo.

10.2. Eliminar CSS no utilizado

Con el tiempo, es común que los archivos CSS acumulen estilos que ya no se usan, aumentando su peso y reduciendo el rendimiento. Herramientas como **PurgeCSS** te ayudan a limpiar estas reglas inútiles.

```
1  const purgecss = require('@fullhuman/postcss-purgecss')({
2      content: ['./src/**/*.html'],
3      defaultExtractor: content => content.match(/[\w-/:]+(?:\s|:)/g) || []
4  });
```

10.3. Usar el cargado condicional de CSS

El uso de carga condicional de CSS puede mejorar los tiempos de carga al cargar solo los estilos necesarios cuando se requieren. Puedes aplicar este enfoque cargando archivos CSS específicos solo para ciertas páginas o componentes.

```
1 <!-- Estilos que solo se cargarán en dispositivos móviles -->
2 <link rel="stylesheet" href="mobile.css" media="only screen and (max-width:
  768px)">
```

10.4. Uso de variable CSS para consistencia

Las variables CSS permite definir valores reutilizables como colores, tamaños o fuentes, lo que reduce la repetición y hace que el CSS sea más fácil de mantener y optimizar.

```
1 :root {
2   --primary-color: #3498db;
3   --font-size-base: 16px;
4 }
5
6 body {
7   color: var(--primary-color);
8   font-size: var(--font-size-base);
9 }
```

10.5. Combinar archivos CSS

Tener muchos archivos CSS pequeños puede aumentar el número de solicitudes HTTP, lo que puede afectar el rendimiento del sitio. Combinar varios archivos

CSS en uno solo puede ayudar a reducir el número de solicitudes y mejorar la velocidad de carga.

```
1  const gulp = require('gulp');
2  const concat = require('gulp-concat');
3
4  gulp.task('combine-css', function() {
5    return gulp.src('src/css/*.css')
6      .pipe(concat('all.css'))
7      .pipe(gulp.dest('dist/css'));
8  });
```

10.6. Uso eficiente de los media queries

Las media queries son esenciales para crear diseños responsivos. Sin embargo, es importante usarlas de manera eficiente para evitar duplicar reglas de estilo innecesarias.

```
1  /* Estilos generales */
2  body {
3    font-size: 16px;
4  }
5  /* Media queries agrupadas */
6  @media (max-width: 768px) {
7    body {
8      font-size: 14px;
9    }
10 }
11 @media (max-width: 480px) {
12   body {
13     font-size: 12px;
14   }
15 }
```

10.7. Prefetching y prerendering de CSS

El prefetching y prerendering permiten que el navegador cargue recursos que el usuario podría necesitar en el futuro, mejorando la percepción

de velocidad en el sitio web. Puedes utilizar estas técnicas para preparar el CSS de las páginas a las que el usuario podría acceder más adelante.

```
1 <!-- Prefetch CSS para futuras navegaciones -->
2 <link rel="prefetch" href="next-page-styles.css">
```

10.8. Reducción del uso de selectores complejos.

Los selectores CSS complejos pueden ralentizar el procesamiento del archivo CSS, ya que el navegador tiene que hacer coincidir estos selectores con los elementos correspondientes en el DOM.

```
1 /* Evitar selectores complejos */
2 div ul li a {
3     color: blue;
4 }
5
6 /* Usar selectores más simples */
7 a {
8     color: blue;
9 }
```

Herramientas recomendadas

A lo largo del desarrollo de esta monografía se ha identificado una serie de herramientas útiles que facilitan el aprendizaje, depuración y aplicación efectiva de CSS avanzado:

1. Visual Studio Code

Editor de código ligero y extensible, con soporte para CSS, autocompletado, previsualización y plugins como *Live Server* y *Prettier*.

2. MDN Web Docs

Documentación oficial y actualizada sobre CSS y otras tecnologías web. Contiene guías, ejemplos y referencias completas.

<https://developer.mozilla.org>

3. **Can I use**

Herramienta que muestra la compatibilidad de propiedades CSS con los distintos navegadores.

<https://caniuse.com>

4. **CodePen / JSFiddle**

Plataformas para crear y compartir prototipos de código HTML, CSS y JavaScript en línea. Son ideales para practicar y experimentar con propiedades avanzadas.

5. **Chrome DevTools**

Consola de desarrollador integrada en Google Chrome, útil para inspeccionar elementos, probar cambios en CSS en tiempo real y analizar el rendimiento de estilos aplicados.

6. **PostCSS / Sass**

Preprocesadores y herramientas que extienden las capacidades de CSS, permitiendo escribir código más limpio y mantenible con variables, mixins y funciones.

7. **A11y Color Contrast Checker**

Herramienta para validar el contraste de colores, garantizando accesibilidad visual según las normas WCAG.

Conclusión

El estudio del CSS avanzado demuestra que el diseño web moderno va mucho más allá de la simple decoración de páginas HTML. Herramientas como variables CSS, funciones como clamp() y calc(), y técnicas avanzadas de maquetación con Flexbox y Grid permiten construir interfaces altamente adaptables, accesibles y eficientes. El conocimiento profundo del modelo de caja, el posicionamiento, y la interacción entre propiedades visuales como filter, clip-path o mask permite a los desarrolladores optimizar tanto la estética como el rendimiento de los sitios web.

Además, el uso de consultas de contenedor (container queries) y scroll snap marcan una evolución hacia diseños que se adaptan no solo al tamaño del dispositivo, sino también al contexto y a la jerarquía del contenido. Estas capacidades son esenciales para la creación de experiencias web modernas que cumplan con los estándares actuales de usabilidad y accesibilidad.

Dominar CSS avanzado no solo mejora las capacidades técnicas de un desarrollador, sino que también le permite participar activamente en el desarrollo de interfaces ricas, responsivas y centradas en el usuario. Esto es clave para destacar en un mercado tecnológico cada vez más competitivo.

Referencias

- [1] *Calculos matemáticos en CSS - CSS en español.* (n.d.). <https://lenguajecss.com/css/funciones-css/funcion-calc/>
- [2] Cagigao, A. M. N., & Mur, A. (2020, August 2). *Introducción a las funciones Min, Max, y Clamp de CSS.* Mamutlove. <https://mamutlove.com/blog/funciones-min-max-clamp-en-css/>
- [3] Coyier, C. (2021). CSS Scroll Snap. CSS-Tricks. <https://css-tricks.com/practical-css-scroll-snapping/>
- [4] Frain, B. (2015). *Responsive Web Design with HTML5 and CSS3* (2nd ed.). Packt Publishing.
- [5] Keith, J. (2010). *HTML5 for Web Designers.* A Book Apart.

[6] Marcotte, E. (2011). Responsive Web Design. A Book Apart.

[7] *min()* - CSS / MDN. (2025, March 10). MDN Web Docs.
<https://developer.mozilla.org/es/docs/Web/CSS/min#sintaxis>

[8] *Uso de propiedades personalizadas (variables) en CSS* - CSS / MDN.
(2025, February 13). MDN Web Docs.
https://developer.mozilla.org/es/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties