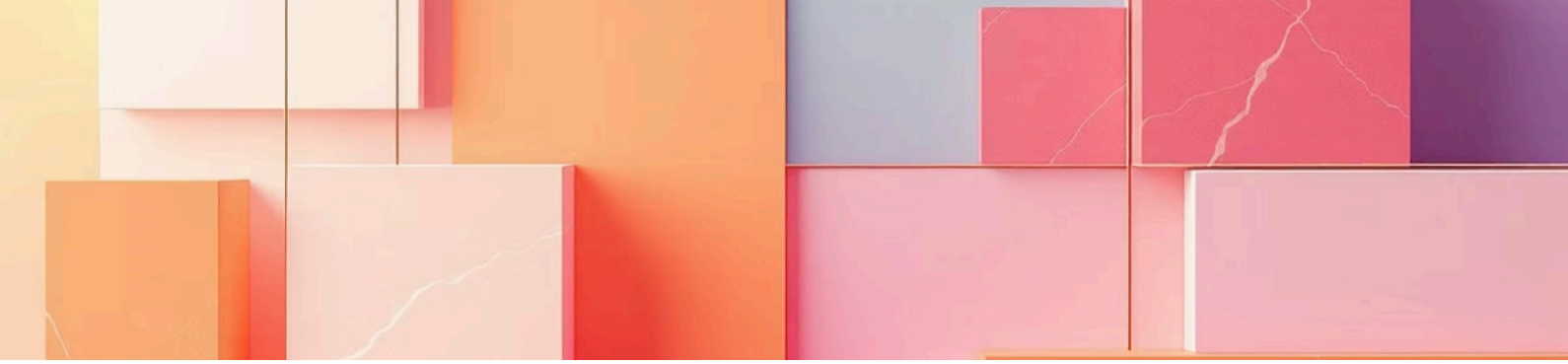
PROMETEO

# Unidad 4:

# Maquetación moderna: Flex y Grid

Lenguaje de marcas y sistemas de gestión de información

Técnico Superior de DAM / DAW



Sesión 16 – Flexbox: ejes, dirección y alineación de elementos

# La base moderna del diseño unidimensional en CSS

Flexbox, o Flexible Box Layout, es uno de los avances más significativos en la historia del diseño web. Antes de su aparición, los diseñadores y desarrolladores dependían de soluciones poco elegantes —como tablas, float o posicionamiento absoluto— para alinear elementos, lo que generaba layouts frágiles, difíciles de mantener y poco responsivos.

Flexbox cambió todo eso introduciendo un modelo unidimensional que permite organizar los elementos en filas o columnas con una flexibilidad sin precedentes. Cuando declaras `display: flex;` en un contenedor, todos sus elementos hijos se convierten en flex items, y este contenedor pasa a tener dos ejes de referencia:

### Eje principal (main axis)

El eje sobre el cual se distribuyen los elementos (por defecto, en fila horizontal).

### Eje secundario (cross axis)

El eje perpendicular al principal, que se usa para alinear verticalmente.

La verdadera potencia de Flexbox radica en su capacidad para adaptarse dinámicamente al espacio disponible, reorganizando y alineando los elementos según las reglas que definamos. Veamos las propiedades clave:

### flex-direction

Define la dirección del eje principal. Puede tomar los valores:

- **row** (por defecto): los elementos se colocan en una fila de izquierda a derecha.
- **row-reverse**: fila en orden inverso (de derecha a izquierda).
- **column**: apila los elementos de arriba abajo.
- **column-reverse**: apila los elementos de abajo hacia arriba.

### justify-content

Controla la distribución de los elementos a lo largo del eje principal. Algunos de sus valores más usados son:

- **flex-start**: los elementos se agrupan al inicio del eje.
- **flex-end**: se agrupan al final.
- **center**: se centran en el eje principal.
- **space-between**: distribuye los elementos con espacio igual entre ellos.
- **space-around o space-evenly**: añade espacio también a los bordes del contenedor.

### align-items

Se usa para alinear los elementos en el eje secundario. Por ejemplo:

- **flex-start**: alinea los items al inicio del eje secundario (arriba si la dirección es horizontal).
- **center**: centra verticalmente (muy usado).
- **flex-end**: los alinea al final.
- **stretch**: los expande para llenar el contenedor.

En combinación, estas propiedades permiten crear estructuras que antes requerían decenas de líneas de CSS. Un ejemplo clásico es centrar un elemento tanto vertical como horizontalmente:

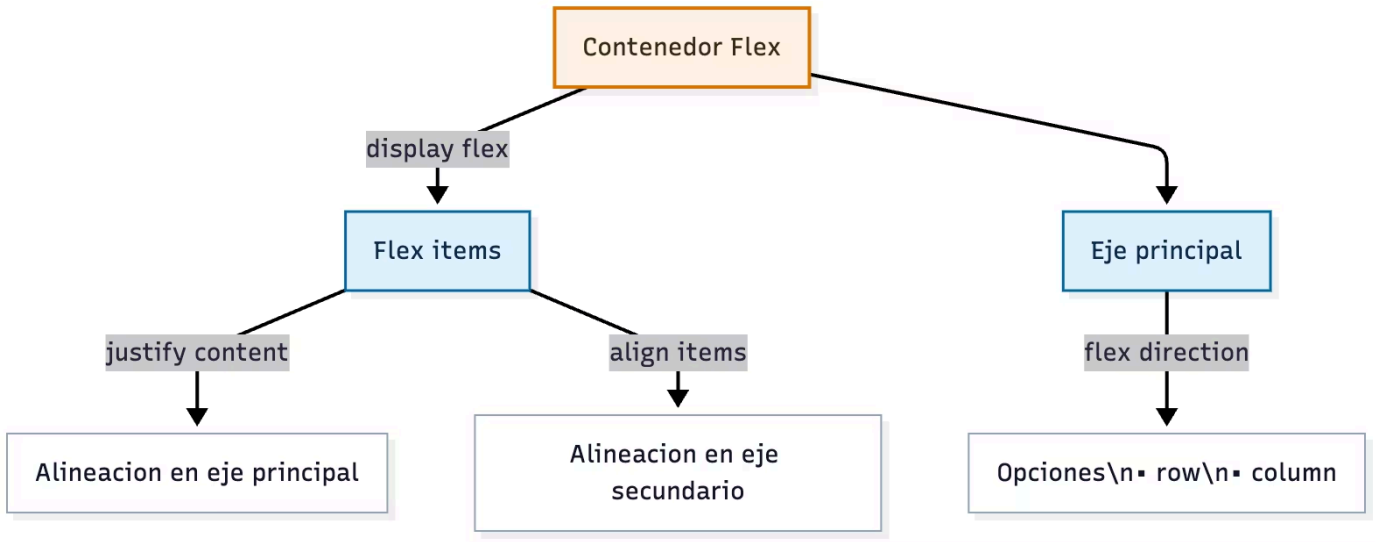
```
.contenedor {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Este código —solo dos líneas— resuelve un problema que durante años fue un dolor de cabeza para los diseñadores. Además, la propiedad `gap` (introducida en Flexbox moderno) permite establecer espacio entre los items sin recurrir a márgenes individuales, simplificando enormemente la consistencia del espaciado.

Flexbox no solo optimiza la estética: mejora la mantenibilidad del código, la responsividad y la accesibilidad del diseño, adaptándose automáticamente a diferentes resoluciones y tamaños de pantalla.

# Esquema Visual: Ejes, dirección y alineación en Flexbox

A continuación se describe el esquema conceptual del funcionamiento de Flexbox y sus propiedades principales.

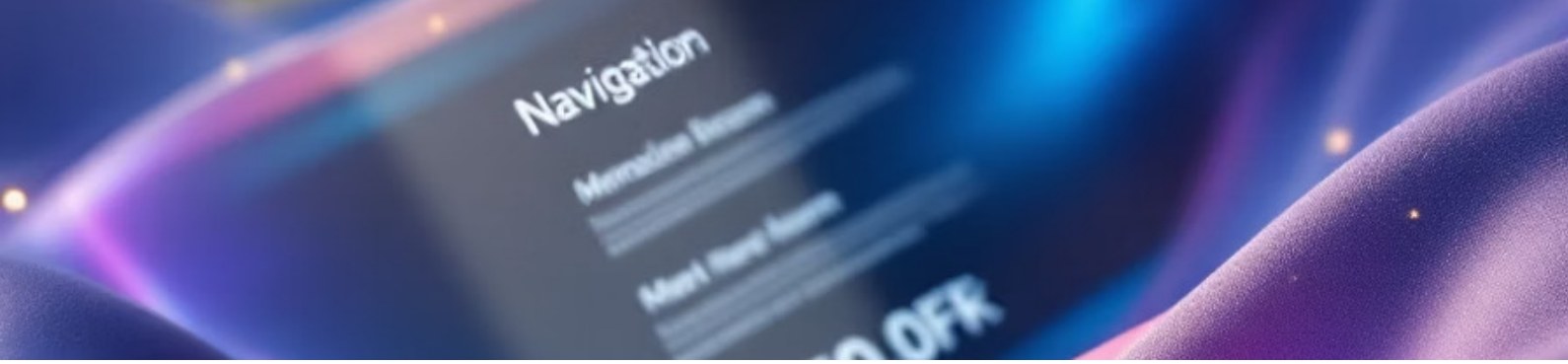



Este esquema ilustra cómo Flexbox introduce una estructura visual clara y modular para distribuir y alinear elementos dentro de cualquier layout.

## Descripción del esquema:

1. **Contenedor Flex (A):** Es el elemento al que se aplica display: flex;.
   Este actúa como el entorno que define cómo se distribuyen sus hijos.
2. **Flex Items (B):** Los hijos directos del contenedor. Se alinean y distribuyen siguiendo las reglas definidas por flex-direction, justify-content y align-items.
3. **Eje Principal (C):** Determina la dirección de los elementos (row o column).
   - Con row, el eje principal es horizontal.
   - Con column, el eje principal es vertical.
4. **Alineación en el Eje Principal (E):** Controlada por justify-content, que gestiona cómo se distribuye el espacio disponible entre los elementos.
5. **Alineación en el Eje Secundario (F):** Controlada por align-items, define la posición de los elementos en el eje perpendicular (vertical si el eje principal es horizontal, y viceversa).

Este esquema ilustra cómo Flexbox introduce una estructura visual clara y modular para distribuir y alinear elementos dentro de cualquier layout.




# Caso de Estudio: Las barras de navegación modernas

Uno de los usos más extendidos y reconocibles de Flexbox se encuentra en las barras de navegación de sitios web y aplicaciones.

## Contexto:

Antes de Flexbox, diseñar una barra de navegación requería técnicas poco elegantes: usar tablas, `float: left;`, o incluso `position: absolute;` con márgenes manuales. Estas soluciones eran difíciles de mantener y se rompían con facilidad en pantallas pequeñas.

## Estrategia:

Flexbox simplifica este diseño a unas pocas líneas. Ejemplo de una barra con un logo a la izquierda y un menú a la derecha:

```
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 30px;
}
```

> **MiMarca**
>
> - Inicio
> - Servicios
> - Contacto

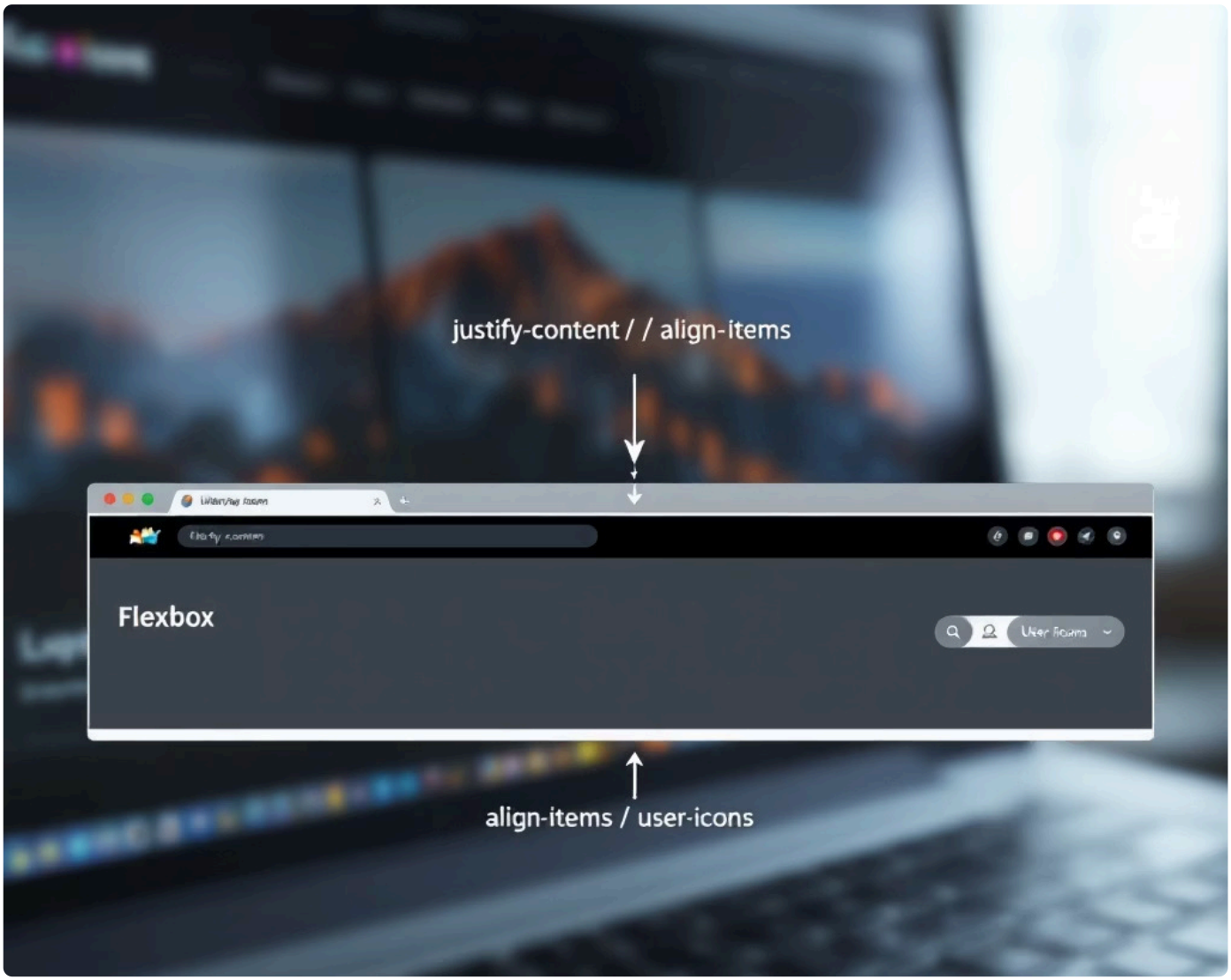
## Resultado:

Con `justify-content: space-between;`, el logo se empuja automáticamente al extremo izquierdo y el menú al derecho, distribuyendo el espacio intermedio de forma uniforme. Además, `align-items: center;` mantiene todo alineado verticalmente sin necesidad de ajustes adicionales.

### Impacto real:

Empresas como Shopify, Spotify o Netflix utilizan Flexbox en sus cabeceras por su capacidad de adaptarse a múltiples dispositivos sin escribir CSS adicional. En el rediseño de la interfaz de Netflix Web, por ejemplo, Flexbox permitió alinear dinámicamente el logo, el menú y los iconos de usuario en una sola línea, garantizando consistencia en todos los tamaños de pantalla, desde un televisor hasta un móvil.

Flexbox, en definitiva, ha hecho que la maquetación responsive sea mucho más limpia, predecible y eficiente.



# Herramientas y Consejos para tu Futuro Profesional

## Flexbox Froggy (https://flexboxfroggy.com/)

Es una herramienta interactiva en forma de juego donde "riegas nenúfares" usando propiedades de Flexbox. En apenas 20 minutos puedes dominar justify-content, align-items, flex-direction y otras propiedades clave.

## Diferencia entre justify-content y align-items:

Es fundamental entender que trabajan en ejes distintos.

- **justify-content** = alineación horizontal (eje principal).
- **align-items** = alineación vertical (eje secundario).

Recordarlo evitará el 80 % de los errores típicos en Flexbox.

## 🪄 Usa gap para el espaciado interno:

En lugar de asignar margin a cada elemento hijo, utiliza gap en el contenedor:

```
.contenedor {
  display: flex;
  gap: 20px;
}
```

Esto garantiza un espacio uniforme y mejora la legibilidad del código.

## Inspecciona con DevTools:

Los navegadores modernos (como Chrome y Firefox) visualizan los ejes y el espaciado de Flexbox directamente. Al inspeccionar un contenedor display: flex;, verás líneas de colores que indican la distribución y alineación de los elementos.

## Flexbox y la Responsividad:

Usa `flex-wrap: wrap;` para que los elementos salten a una nueva línea cuando no haya espacio suficiente. Esto evita desbordamientos y garantiza layouts adaptables sin necesidad de media queries complejas.

# Mitos y Realidades

### ❌ Mito: "Flexbox es solo para centrar cosas."

→ **FALSO.** Aunque es cierto que centrar elementos con Flexbox (justify-content: center; align-items: center;) es una de sus aplicaciones más populares, su potencia va mucho más allá. Flexbox es un sistema completo de distribución y alineación que permite controlar la dirección, el espaciado y la proporción del tamaño de los elementos en una dimensión.

### ❌ Mito: "Flexbox es difícil de aprender."

→ **FALSO.** Flexbox puede parecer complejo al inicio por la cantidad de propiedades, pero su lógica es muy intuitiva una vez se comprenden los conceptos de eje principal y eje secundario. Además, herramientas como Flexbox Froggy o las DevTools lo hacen visual y accesible incluso para principiantes. La mayoría de los diseñadores lo dominan con unas pocas horas de práctica.

---

## Resumen Final

- Flexbox es un modelo unidimensional de CSS para distribuir y alinear elementos.
- Se activa con `display: flex;` en el contenedor.
- `flex-direction` controla el eje principal (fila o columna).
- `justify-content` alinea en el eje principal; `align-items`, en el secundario.
- La propiedad `gap` permite espaciar elementos de forma uniforme.
- Es ideal para layouts lineales como barras de navegación o secciones de tarjetas.

Sesión 17 – Flexbox avanzado: grow, shrink, basis y order

# El control del espacio y el orden en el diseño flexible

Flexbox (Flexible Box Layout) revolucionó la maquetación web al permitir una distribución dinámica del espacio entre elementos dentro de un contenedor, adaptándose a cualquier tamaño de pantalla sin necesidad de cálculos manuales. Una vez que has comprendido cómo alinear y distribuir elementos, el siguiente paso es dominar cómo controlarlos individualmente: su tamaño, proporción y orden visual.

Para ello, existen cuatro propiedades clave aplicadas a los flex items (hijos): `flex-grow`, `flex-shrink`, `flex-basis` y `order`. Estas determinan cómo cada elemento se comporta dentro del espacio disponible del contenedor y en qué posición visual aparece. Veamos cada una con detalle:

## flex-grow

Define la capacidad de un elemento para crecer si hay espacio libre en el contenedor. Por ejemplo, si un contenedor tiene tres elementos con `flex-grow: 1`, todos crecerán de manera equitativa para llenar el espacio restante. Pero si uno tiene `flex-grow: 2` y los otros 1, el primero crecerá el doble que los demás.

Su valor no es una medida en píxeles, sino un factor de proporción. Si el contenedor tiene espacio sobrante, Flexbox distribuye ese espacio entre los elementos en función de la proporción definida por `flex-grow`.

## flex-shrink

Hace lo contrario: define la capacidad del elemento para encogerse cuando no hay suficiente espacio. Si todos los elementos tienen `flex-shrink: 1`, se reducirán por igual. Si uno tiene `flex-shrink: 2`, se encogerá el doble que los demás.

Esto evita que los contenidos se desborden o que el diseño se rompa cuando el viewport se reduce. Un uso común es desactivar el encogimiento estableciendo `flex-shrink: 0`, útil en botones, logotipos o elementos que no deben deformarse.

### flex-basis

Es el tamaño base que tendrá el elemento antes de aplicar los crecimientos o reducciones. Puede definirse en píxeles (`flex-basis: 200px;`), en porcentaje o como `auto`, que tomará el ancho natural del contenido o el valor de `width` si existe.

En términos sencillos: `flex-basis` indica el punto de partida a partir del cual `flex-grow` y `flex-shrink` comienzan a actuar.

### order

Flexbox no solo controla tamaños, sino también el orden visual de los elementos. Con `order`, puedes modificar la posición de un elemento sin cambiar el HTML. Por defecto, todos tienen `order: 0`. Los elementos con menor valor aparecen primero, y los de mayor, después.

Sin embargo, es importante recordar que `order` solo afecta el orden visual, no el orden semántico. Los lectores de pantalla y los motores de búsqueda seguirán el orden real del HTML.

## flex (shorthand)

Estas tres propiedades pueden escribirse en una sola línea: `flex: grow shrink basis;`

Por ejemplo: `flex: 1 1 0%;` significa que el elemento puede crecer (1), encogerse (1) y que su tamaño base es 0%. Un atajo muy usado es `flex: 1;`, equivalente a `flex: 1 1 0%`, lo que hace que los elementos compartan equitativamente el espacio disponible.

Ejemplo de `order`:

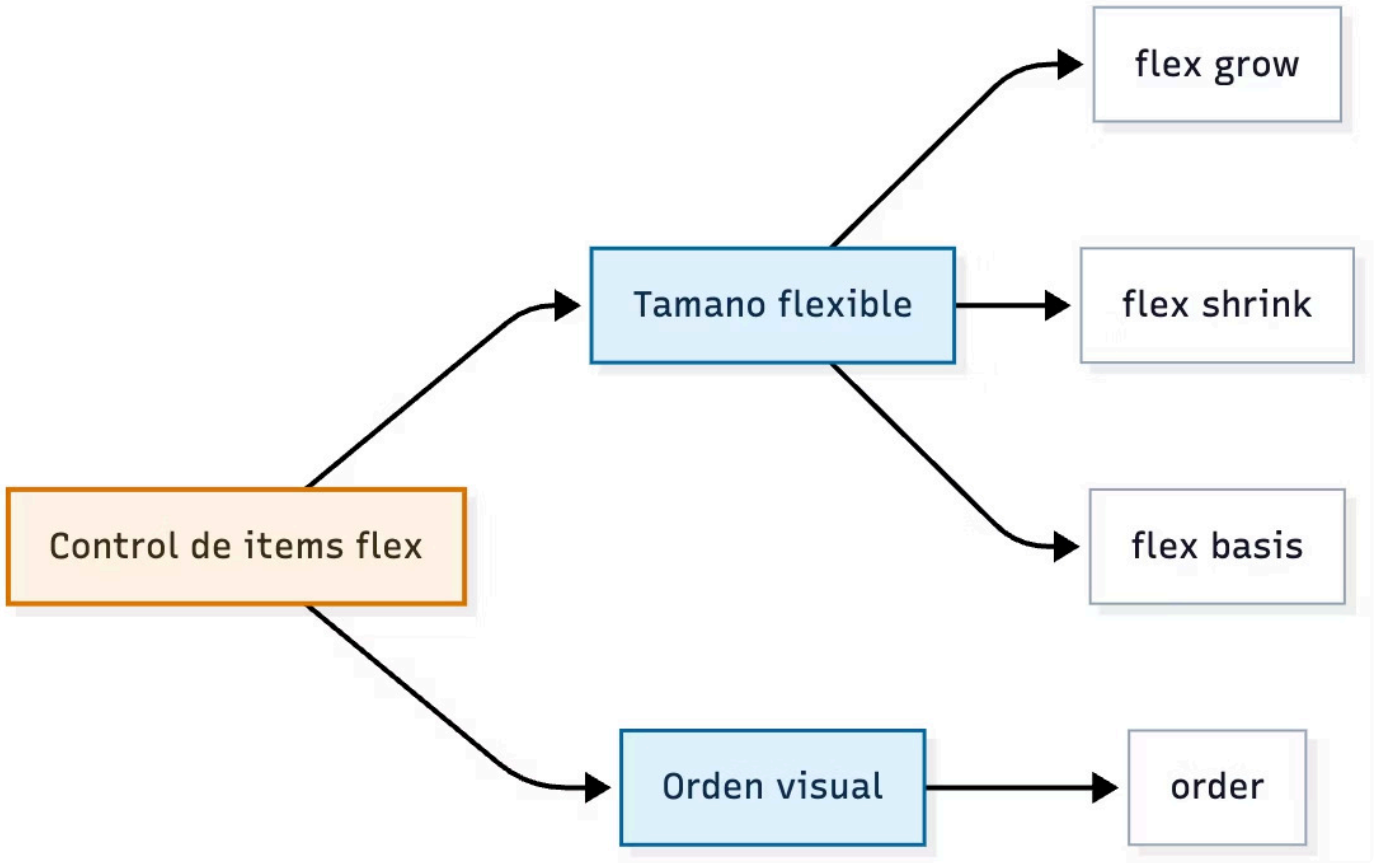
```
.item1 { order: 2; }
.item2 { order: 1; }
```

hará que `.item2` aparezca antes, aunque en el código HTML esté después. Esto resulta útil en diseños responsivos, donde tal vez un menú que en escritorio está al lado derecho, en móvil debe pasar a la parte superior.

## El equilibrio entre flexibilidad y coherencia

Flexbox busca el balance entre libertad y control. Saber usar `grow`, `shrink` y `basis` permite que el diseño sea adaptable sin romper la estructura. Combinados con `order`, convierten cualquier layout en una cuadrícula viva que se reacomoda con inteligencia. En definitiva, Flexbox no solo distribuye espacio: distribuye intención visual.

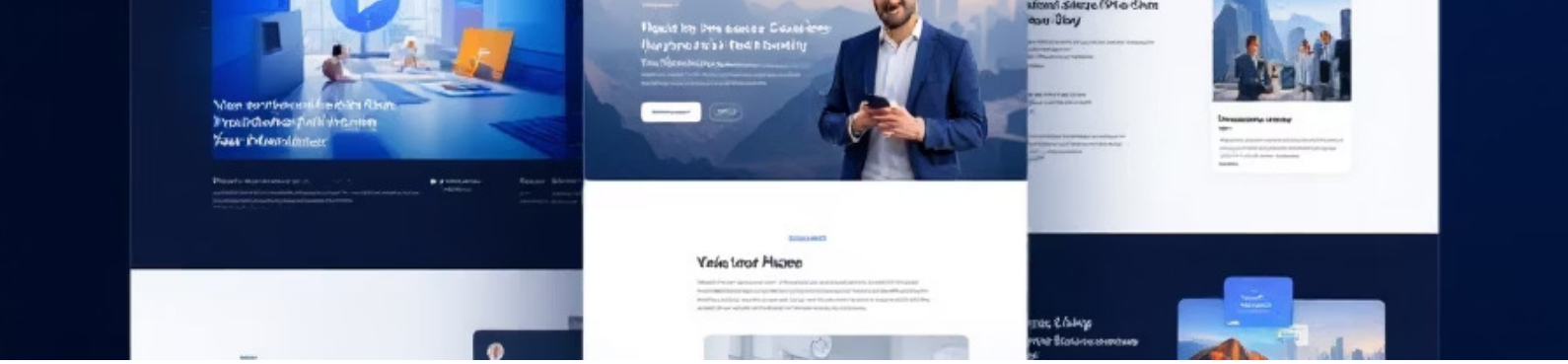
# Esquema Visual: Propiedades avanzadas de Flexbox




**Descripción del esquema:**

- El nodo central Control de Items Flex representa el conjunto de propiedades aplicadas a los hijos del contenedor flex.
- **Tamaño Flexible** agrupa las tres propiedades que definen cómo se distribuye el espacio: flex-grow (capacidad para crecer), flex-shrink (capacidad para encogerse), y flex-basis (tamaño inicial de referencia).
- **Orden Visual** incluye la propiedad order, encargada de definir la posición visual de los elementos.

En conjunto, estas propiedades determinan la estructura final del layout flexible, tanto en tamaño como en disposición visual.



# 3. Caso de Estudio: El "Holy Grail Layout" y el control del espacio

Durante años, los diseñadores web intentaron reproducir un mismo patrón de diseño: cabecera, pie, contenido principal y dos barras laterales, conocido como el Holy Grail Layout (o "Diseño del Santo Grial"). En la era previa a Flexbox, lograrlo requería estructuras HTML complejas, flotados, y cálculos de márgenes difíciles de mantener.

## Contexto

El objetivo era conseguir un layout con tres columnas (contenido principal y dos sidebars), donde la columna central creciera según el espacio disponible, mientras las laterales mantuvieran un ancho fijo. Además, en dispositivos móviles, las barras laterales debían reordenarse: por ejemplo, pasar arriba o abajo del contenido.

## Estrategia

Con Flexbox, esta estructura se simplifica drásticamente:

```
.container {
  display: flex;
}
.main {
  flex-grow: 1;
  flex-shrink: 1;
  flex-basis: 0%;
}
.sidebar {
  flex: 0 0 200px;
}
```

El contenedor reparte el espacio disponible entre los elementos:

- La clase `.main` crece para ocupar todo el espacio sobrante.
- Las `.sidebar` mantienen un ancho fijo de 200px.

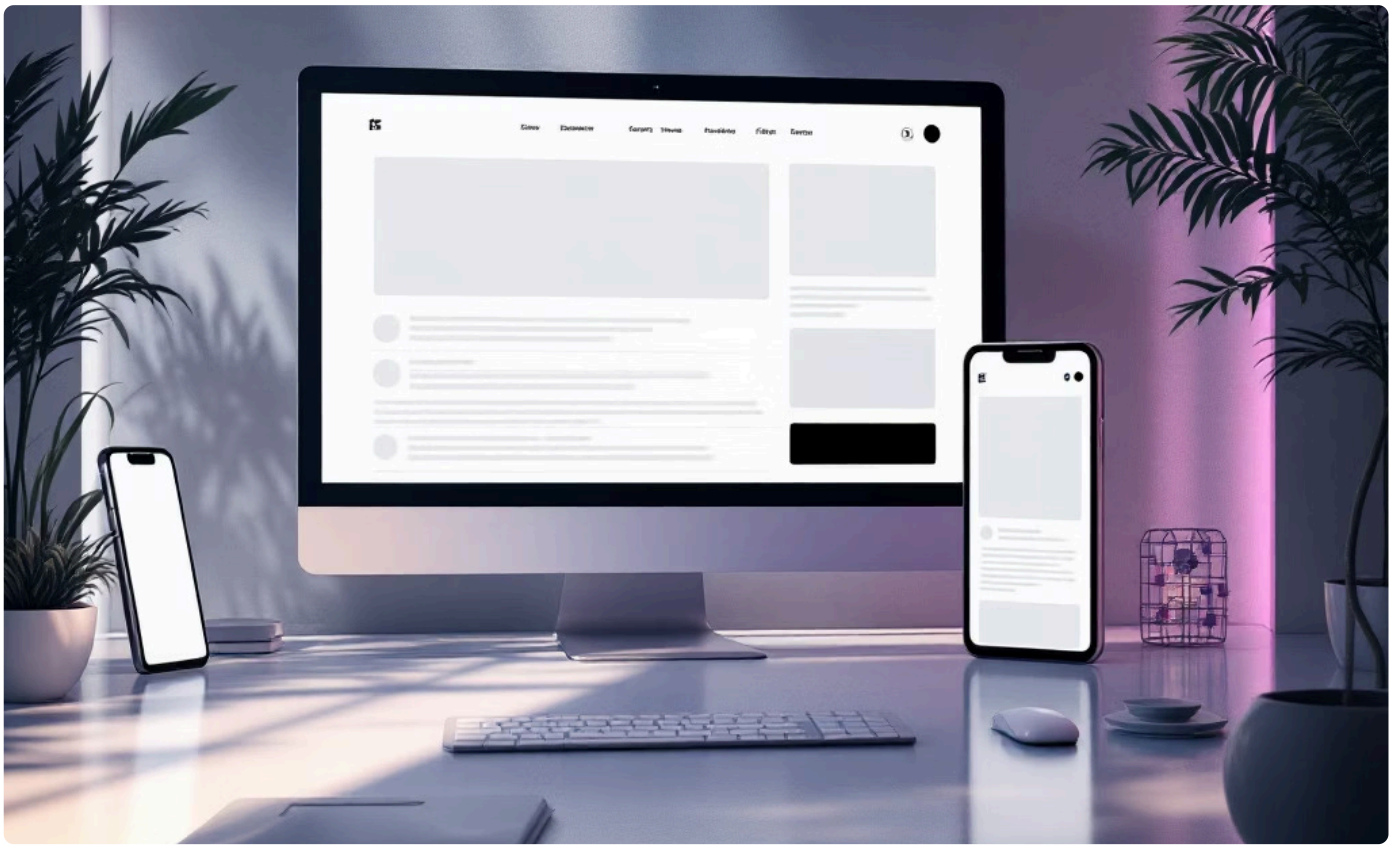
Para adaptarlo a móvil, simplemente se cambia la dirección y el orden:

```
@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
  .sidebar:first-child {
    order: -1;
  }
}
```

Con este simple cambio de `order`, la barra lateral puede pasar al principio sin duplicar el HTML ni reescribir la estructura.

## Resultado

El layout se vuelve dinámico, fluido y semánticamente coherente. El HTML mantiene un orden lógico para accesibilidad y SEO, mientras que el CSS controla la presentación visual. Así, Flexbox permite resolver en pocas líneas un problema histórico de diseño web.



# Herramientas y Consejos

## Domina el atajo flex: 1;

Es una de las combinaciones más prácticas. Al aplicarlo a varios hijos, todos comparten el espacio disponible de forma proporcional, sin tener que definir cada valor por separado. Ideal para menús horizontales o tarjetas en fila.

## Usa order estratégicamente en diseños responsivos

Cambiar el orden visual según el tamaño de pantalla permite una experiencia adaptativa. Por ejemplo, en un layout de producto, el texto puede estar a la derecha en escritorio y debajo en móvil con solo una línea de CSS.

## Evita conflictos entre width y flex-basis

Si defines ambos, `flex-basis` tiene prioridad. Si necesitas que un elemento respete un ancho fijo sin crecer ni encogerse, usa `flex: 0 0 250px;`.

## Visualiza el espacio con DevTools

Los navegadores modernos, como Chrome o Firefox, incluyen herramientas de inspección para contenedores flex. Actívalas y observa cómo se reparte el espacio al modificar grow o shrink. Es la mejor manera de entender cómo Flexbox toma decisiones de distribución.

## Prueba con Flexbox Froggy

Es un juego interactivo gratuito donde puedes practicar todas las propiedades de Flexbox. En pocos niveles entenderás intuitivamente cómo interactúan grow, shrink y order.

## Recuerda el impacto en la accesibilidad

Cambiar el orden visual puede confundir a los lectores de pantalla. Si usas order para reorganizar, asegúrate de que el orden del HTML siga siendo lógico.

# Mitos y Realidades

### ❌ Mito: "flex-grow se mide en píxeles."

→ **FALSO.** flex-grow no mide tamaño, mide proporción. Si dos elementos tienen flex-grow: 1 y flex-grow: 2, el segundo ocupará el doble de espacio sobrante que el primero. No define cuántos píxeles crece, sino qué proporción del espacio libre recibe.

### ❌ Mito: "Cambiar order cambia el orden en el HTML."

→ **FALSO.** order solo afecta al orden visual. El HTML conserva su secuencia original, que es la que interpretan los buscadores y lectores de pantalla. Por ello, debe usarse con cuidado: alterar visualmente el orden sin mantener coherencia semántica puede afectar la accesibilidad.

---

## Resumen Final (para examen)

- `flex-grow`, `flex-shrink` y `flex-basis` controlan el tamaño flexible de los elementos hijos.
- `flex-grow`: capacidad de crecer; `flex-shrink`: capacidad de encogerse.
- `flex-basis`: tamaño base inicial del elemento.
- `order` cambia el orden visual, no el HTML.
- `flex: 1;` es un atajo común para repartir el espacio equitativamente.
- Usar `order` con cuidado evita conflictos de accesibilidad.

## Sesión 18 – Grid: filas, columnas y fracciones

# El poder del diseño bidimensional en CSS

Durante muchos años, los diseñadores web tuvieron que recurrir a soluciones improvisadas para crear maquetaciones complejas. Usar tablas en los años 2000, float en la era del CSS2 o frameworks pesados como Bootstrap eran prácticas comunes porque CSS no ofrecía un sistema nativo para gestionar layouts bidimensionales.

Todo cambió con la llegada de CSS Grid Layout. Grid (rejilla o parrilla) es el primer sistema de maquetación verdaderamente bidimensional en CSS. A diferencia de Flexbox, que se centra en una sola dirección (horizontal o vertical), Grid permite organizar elementos en filas y columnas simultáneamente. Esta capacidad convierte a Grid en la herramienta ideal para diseñar páginas completas, interfaces de aplicaciones, paneles administrativos o dashboards de datos.

### Cómo funciona Grid

Para activar una cuadrícula en CSS, basta con aplicar: `display: grid;` a un contenedor. Todos sus elementos hijos se convierten automáticamente en grid items y pueden ubicarse dentro de las celdas que definas.

El contenedor Grid se estructura mediante líneas, filas y columnas. Tú decides cuántas columnas y filas quieres, y qué proporción de espacio ocupa cada una. Las propiedades clave son:

#### grid-template-columns

Define el número y tamaño de las columnas.

#### grid-template-rows

Define el número y tamaño de las filas.

#### gap

Establece el espacio entre filas y columnas (reemplaza al antiguo grid-gap).

Ejemplo básico:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: auto;
  gap: 20px;
}
```

Este código crea una cuadrícula de tres columnas iguales con un espacio de 20px entre cada una. Los elementos se distribuyen automáticamente en las celdas, llenando la cuadrícula fila a fila.

## La unidad fr (fracción)

La gran revolución de CSS Grid es la introducción de la unidad fr, abreviatura de fraction. Representa una fracción del espacio disponible, y hace que el diseño se adapte automáticamente a cualquier tamaño de pantalla sin necesidad de cálculos con porcentajes o píxeles.

Por ejemplo: `grid-template-columns: 2fr 1fr;` significa que la primera columna ocupará el doble de espacio que la segunda, independientemente del ancho total del contenedor. Esto convierte la maquetación responsive en algo natural y predecible.

## Rejillas declarativas y explícitas

Grid permite definir explícitamente la estructura de una página:

- puedes nombrar líneas y áreas (`grid-template-areas`),
- posicionar elementos de forma precisa (`grid-column` y `grid-row`),
- o dejar que el navegador los coloque automáticamente según el flujo natural.

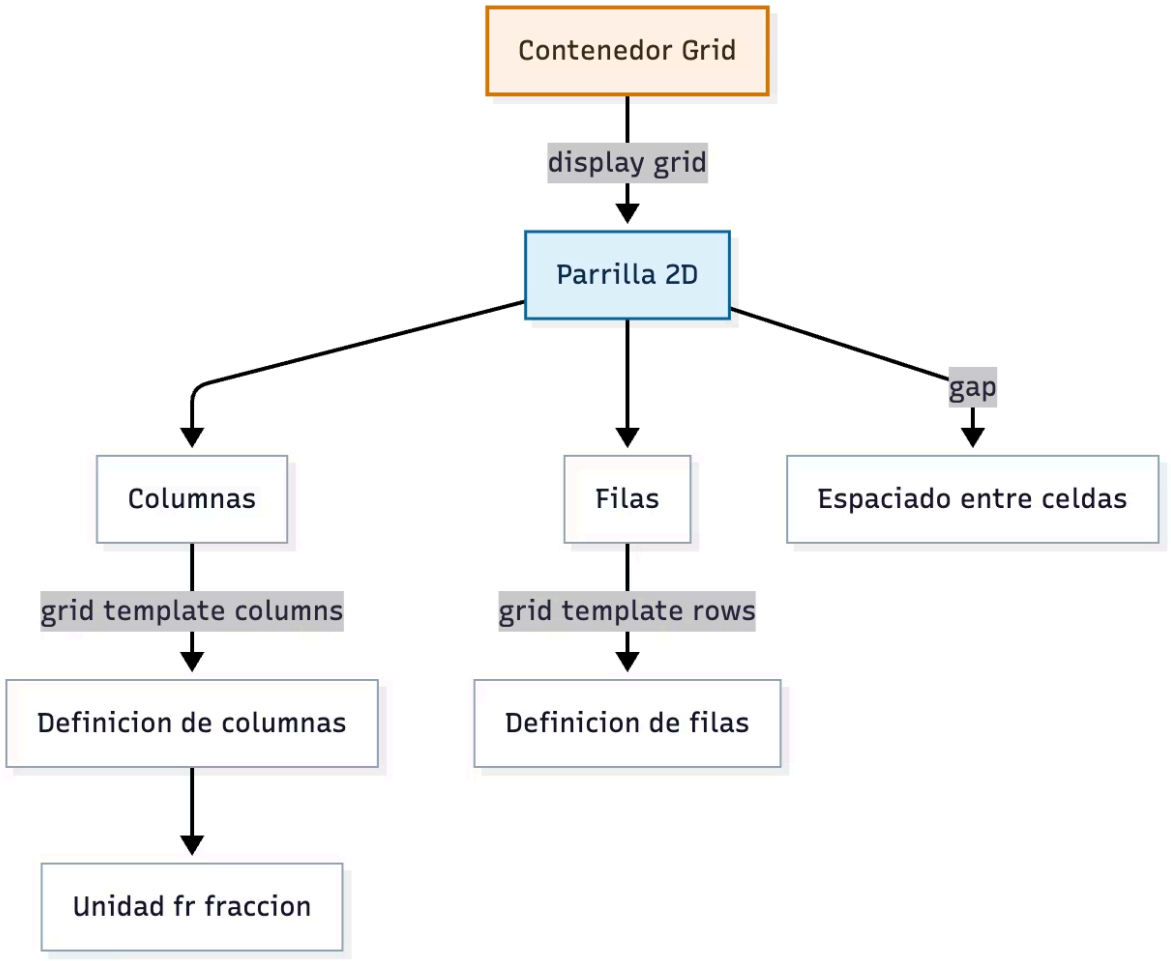
La lógica de Grid es declarativa, es decir: defines el resultado que quieres, y CSS se encarga de calcular las posiciones y tamaños. Esto simplifica el mantenimiento y reduce el código.

## Grid vs Flexbox

Flexbox y Grid no compiten: se complementan. Flexbox: ideal para organizar elementos en una fila o columna (por ejemplo, un menú). Grid: ideal para estructurar toda la página o secciones con filas y columnas (por ejemplo, el cuerpo de un sitio o un panel administrativo).

Una práctica profesional habitual es usar Grid para el esqueleto principal y Flexbox para alinear los componentes internos dentro de cada celda.

# Esquema Visual: Cómo se organiza un Grid




**Descripción detallada del esquema:**

1. **Contenedor Grid (A):** elemento que activa la cuadrícula con display: grid;.
   Define el sistema bidimensional que organiza los hijos.
2. **Parrilla 2D (B):** estructura formada por filas y columnas que crea celdas. Cada celda es un posible espacio para un elemento hijo.
3. **Columnas (C) y Filas (D):** ejes que definen la dirección horizontal y vertical.
   - grid-template-columns controla el ancho de cada columna.
   - grid-template-rows controla la altura de cada fila.
4. **Unidad fr (G):** permite definir las proporciones relativas de espacio sin depender de píxeles o porcentajes.
5. **gap (H):** define la separación entre las celdas, tanto en filas como en columnas, sin necesidad de márgenes individuales.

Este diagrama refleja la lógica visual de Grid: un **mapa de celdas perfectamente alineadas** que puedes controlar de manera precisa, flexible y limpia.

# 3. Caso de Estudio: El rediseño de CSS-Tricks con CSS Grid

## Contexto

CSS-Tricks, uno de los blogs técnicos más importantes del mundo del desarrollo web (fundado por Chris Coyier), fue pionero en adoptar CSS Grid en su rediseño de 2017. Antes de Grid, su estructura de artículos, banners y módulos laterales dependía de frameworks y float, lo que hacía difícil mantener la consistencia entre pantallas.

## Estrategia

El nuevo layout se diseñó completamente con Grid Layout:

- Se definieron 12 columnas con `grid-template-columns: repeat(12, 1fr);`.
- Las áreas principales (cabecera, contenido, barra lateral y pie) se posicionaron mediante `grid-template-areas`.
- Los artículos destacados ocuparon múltiples columnas y filas, logrando un efecto de mosaico similar a una revista digital.

El código base era sorprendentemente simple:

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
  grid-gap: 20px;
  grid-template-areas:
    "header header header header header header header header header header header header"
    "main main main main main main main main sidebar sidebar sidebar sidebar"
    "footer footer footer footer footer footer footer footer footer footer footer footer";
}
```

## Resultado

### Flexibilidad

La estructura se adaptaba automáticamente a cualquier resolución, proporcionando una experiencia de usuario fluida.

### Mantenibilidad

Se logró una reducción del 40% en el CSS original, simplificando significativamente el código base.

### Accesibilidad visual

El contenido se alineaba de forma coherente y proporcional en todos los dispositivos, mejorando la legibilidad.

### Innovación

El rediseño se convirtió en un referente de buenas prácticas y un modelo a seguir en el diseño web moderno.

## Impacto real

Tras la implementación de Grid, la comunidad de CSS-Tricks reportó una reducción del tiempo de desarrollo de prototipos y una mejora significativa en la coherencia visual de los diseños. Desde entonces, empresas como Mozilla, Smashing Magazine y Web.dev adoptaron estructuras similares basadas en Grid para sus interfaces.

# Herramientas y Consejos para tu Futuro Profesional

## Grid Garden (https://cssgridgarden.com/)

El equivalente a Flexbox Froggy, pero para Grid. Es un juego interactivo en el que "riegas zanahorias" resolviendo desafíos que te enseñan propiedades como grid-column, grid-row o grid-template-areas. Aprendes jugando y memorizas de manera natural.

## Usa las DevTools del navegador

Todos los navegadores modernos (Chrome, Firefox, Edge) incluyen herramientas de visualización de Grid. En Firefox, por ejemplo, puedes activar la vista de "líneas de cuadrícula" para ver exactamente cómo se distribuyen las filas y columnas. Esto es vital para depurar problemas de alineación.

## Aprovecha la función repeat()

Evita escribir código repetitivo. En lugar de escribir:

```
grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr;
```

puedes usar:

```
grid-template-columns: repeat(6, 1fr);
```

Esto mejora la legibilidad y la eficiencia.

## 🪄 Combina Grid con unidades mixtas (px, %, fr)

Un diseño profesional suele usar una combinación de unidades:

```
grid-template-columns: 200px 1fr 2fr;
```

Aquí defines una columna fija para el menú lateral, una adaptable para el contenido y otra más amplia para el bloque principal.

## Usa auto-fit y minmax() para diseño responsive

Estas funciones hacen que el Grid se adapte automáticamente al espacio disponible:

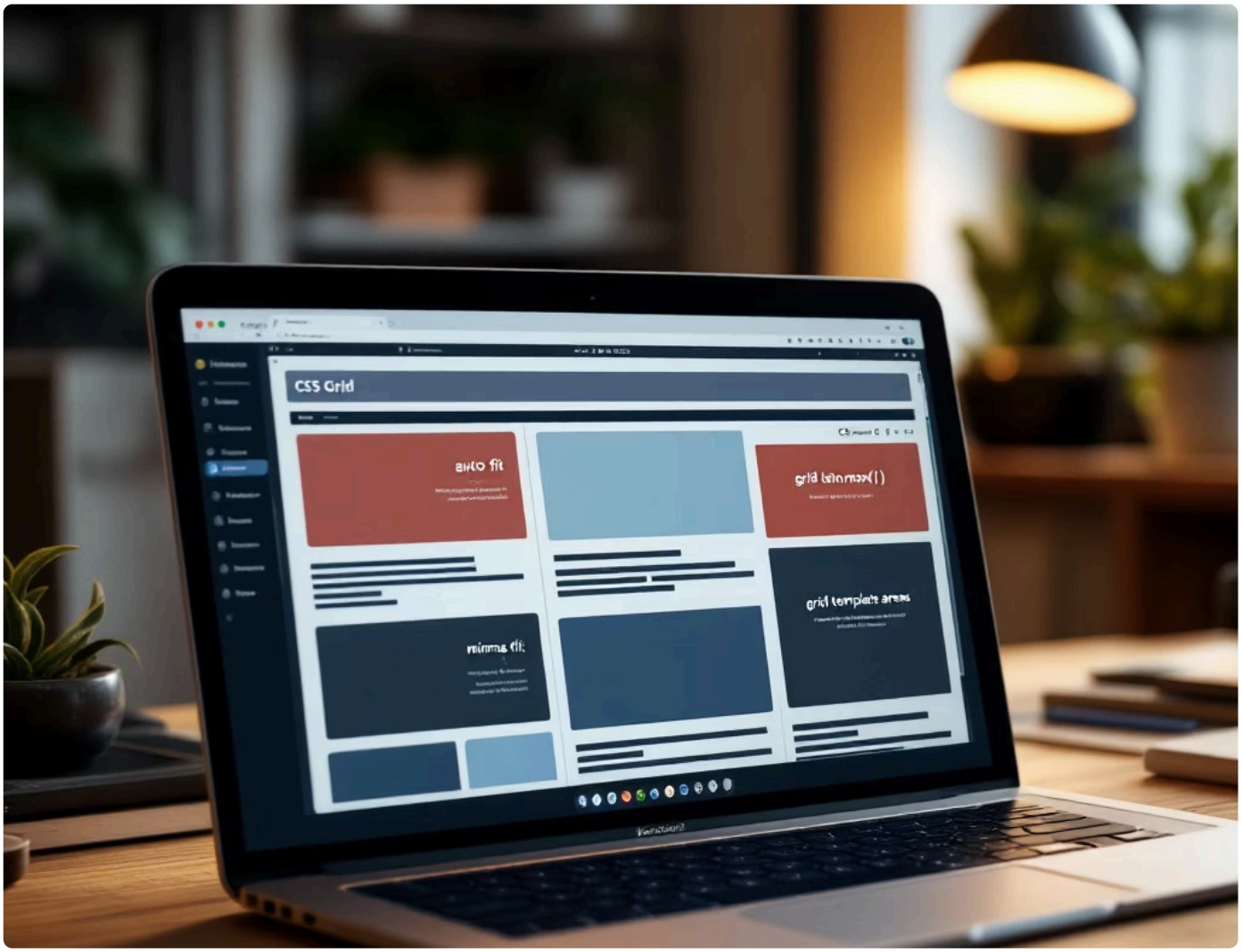
```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

Con esto puedes crear galerías responsivas donde los elementos se redistribuyen automáticamente según el ancho de la pantalla.

## Planifica tus áreas de contenido con grid-template-areas

Nombrar las zonas de tu Grid mejora la claridad del código y facilita la comunicación con otros diseñadores o desarrolladores. Ejemplo:

```
grid-template-areas:
 "header header"
 "sidebar main"
 "footer footer";
```



# Mitos y Realidades

### ❌ Mito: "CSS Grid reemplaza a Flexbox."

→ **FALSO.** Grid y Flexbox están diseñados para resolver problemas distintos. Flexbox es ideal para la alineación unidimensional (una fila o columna). Grid, en cambio, gestiona layouts bidimensionales, estructurando toda la página o secciones con múltiples filas y columnas. En la práctica, ambos se combinan: Grid para la estructura general y Flexbox para los detalles internos.

### ❌ Mito: "Grid es demasiado complicado para layouts simples."

→ **FALSO.** Aunque Grid puede crear maquetaciones muy complejas, también simplifica las más básicas. Por ejemplo:

```
display: grid;
grid-template-columns: 1fr 1fr;
gap: 20px;
```

es más limpio y robusto que usar float o inline-block. Grid te da control total con menos código, incluso en diseños minimalistas.

---

## Resumen Final

- CSS Grid es el sistema de maquetación bidimensional de CSS (filas y columnas).
- Se activa con `display: grid;`.
- Las propiedades principales son:
- `grid-template-columns` → define el ancho de las columnas.
- `grid-template-rows` → define la altura de las filas.
- `gap` → crea espacio entre celdas.
- La unidad `fr` representa una fracción del espacio disponible.
- `repeat()` y `minmax()` facilitan diseños modulares y responsivos.
- Grid no sustituye a Flexbox: lo complementa en layouts complejos.

## Sesión 19 – Grid avanzado: áreas, auto-fit, auto-fill y layouts responsive

# El diseño declarativo y responsivo llevado al máximo

Hasta ahora has visto cómo CSS Grid permite crear una parrilla bidimensional para organizar elementos en filas y columnas. Pero el verdadero potencial de Grid se revela cuando aplicas sus características avanzadas: `grid-template-areas`, `auto-fit`, `auto-fill` y la función `minmax()`. Estas herramientas hacen posible diseñar layouts completamente flexibles, legibles y adaptativos, sin apenas usar media queries.

### grid-template-areas: maquetación semántica y visual

Con `grid-template-areas`, puedes nombrar las zonas de tu parrilla y luego asignar elementos a esas zonas por nombre, en lugar de por coordenadas. Esto convierte el código CSS en algo casi visual, parecido a un "dibujo" del layout.

Ejemplo:

```
.container {
  display: grid;
  grid-template-columns: 200px 1fr;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
}

header { grid-area: header; }
aside { grid-area: sidebar; }
main { grid-area: main; }
footer { grid-area: footer; }
```

En este ejemplo, puedes leer la estructura directamente: un encabezado arriba, contenido dividido entre barra lateral y cuerpo principal, y un pie que abarca toda la parte inferior. Lo más potente es que puedes reorganizar el layout entero en una media query simplemente cambiando las líneas de texto dentro de `grid-template-areas`, sin tocar el HTML ni los nombres de clase.

Por ejemplo, para móviles:

```
@media (max-width: 600px) {
  .container {
    grid-template-areas:
      "header"
      "main"
      "sidebar"
      "footer";
  }
}
```

Con una sola modificación textual, todo el diseño se reordena verticalmente. Ninguna otra técnica en CSS ofrece tanta claridad y control.

## auto-fit, auto-fill y minmax(): rejillas fluidas sin media queries

Si `grid-template-areas` te da control semántico, las funciones `auto-fit` y `auto-fill`, combinadas con `minmax()`, te dan adaptabilidad total.

### La función repeat()

Permite repetir patrones en las columnas o filas:

```
grid-template-columns: repeat(4, 1fr);
```

crea cuatro columnas iguales.

### La función minmax(min, max)

Define un tamaño mínimo y máximo:

```
minmax(200px, 1fr)
```

significa: "cada columna debe tener al menos 200px, pero puede crecer hasta ocupar una fracción del espacio disponible".

### auto-fit y auto-fill

Ambas generan automáticamente tantas columnas como quepan dentro del contenedor, pero hay una diferencia sutil:

- **auto-fill**: mantiene el espacio reservado para las columnas aunque no haya contenido suficiente.
- **auto-fit**: colapsa las columnas vacías, haciendo que las existentes crezcan para ocupar todo el espacio.

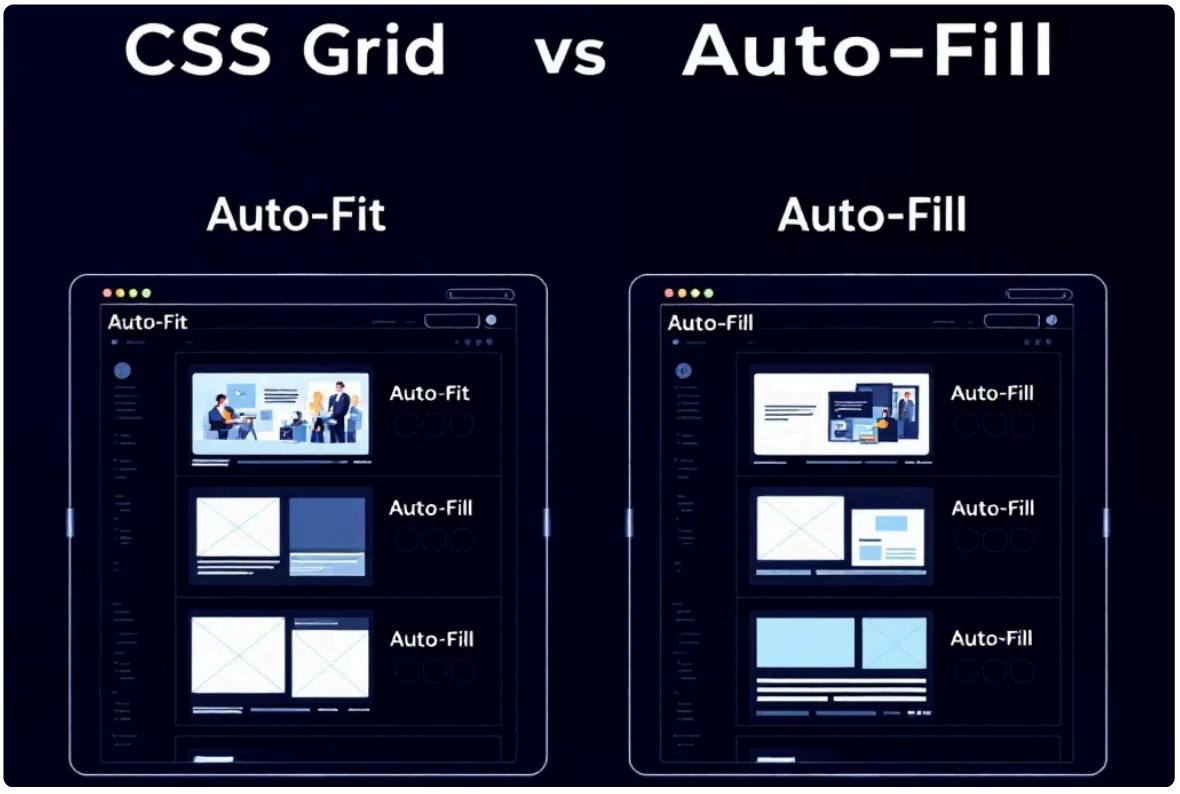
Por ejemplo:

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

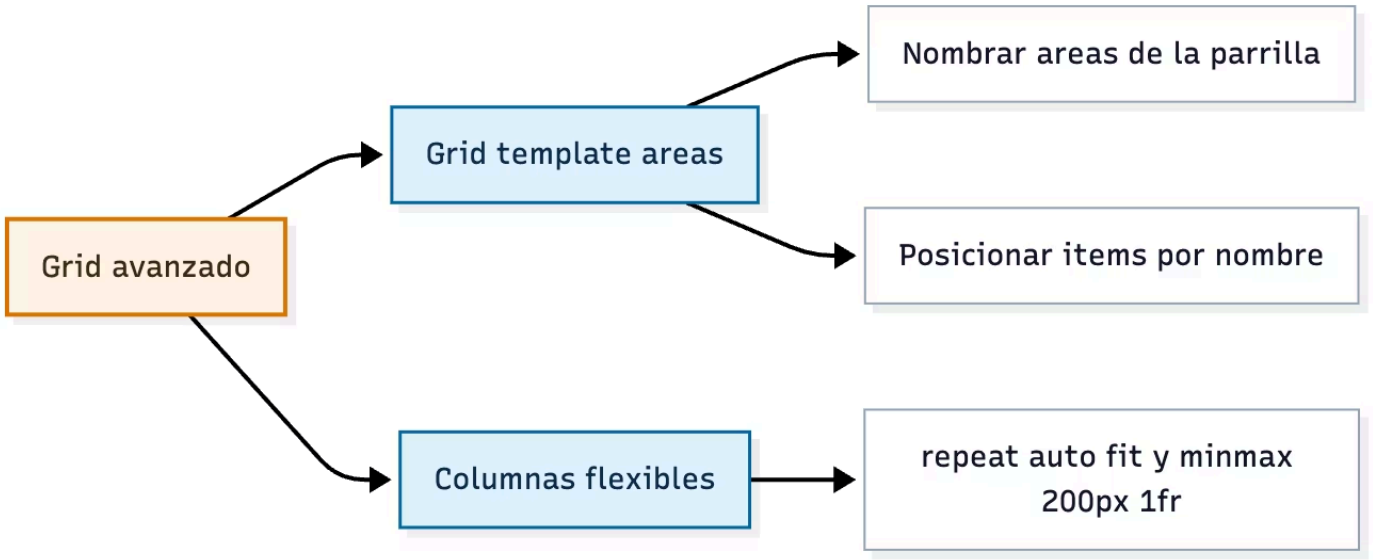
creará una galería flexible que añade o elimina columnas automáticamente según el ancho del contenedor. Esto permite construir layouts 100 % responsivos sin necesidad de media queries. El navegador se encarga de redistribuir los elementos.

## Grid avanzado = control total con menos código

Combinar estas propiedades te da un control declarativo del layout: defines el qué y el navegador resuelve el cómo. Esto no solo simplifica el mantenimiento del CSS, sino que también mejora la legibilidad, accesibilidad y semántica del código. El resultado son layouts profesionales, modernos y fácilmente escalables.



# Esquema Visual: Las dos dimensiones del Grid avanzado

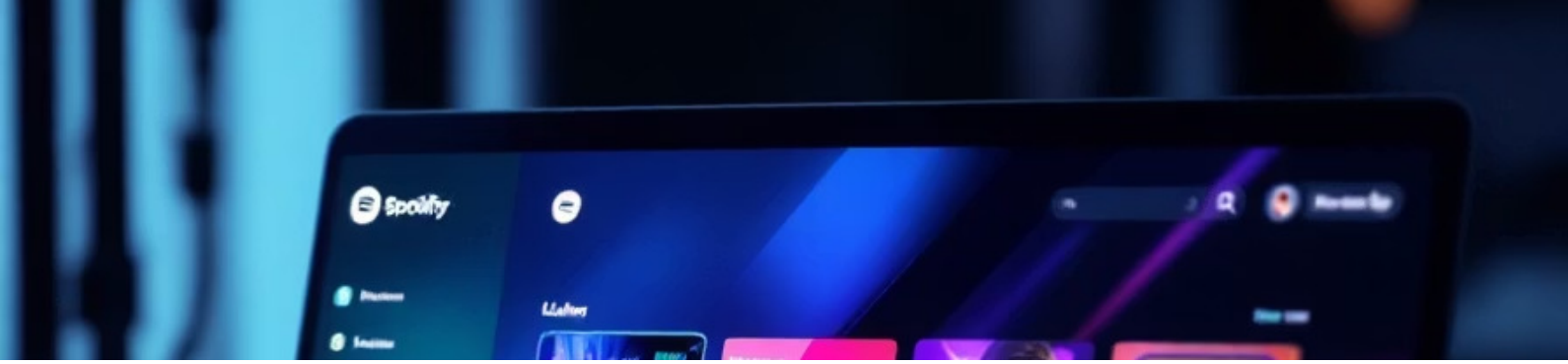



## Descripción del esquema:

1. **Grid Avanzado (A):** representa la evolución del modelo Grid básico hacia uno declarativo y adaptable.
2. **Grid Template Areas (B):** permite nombrar y definir visualmente zonas dentro de la cuadrícula.
   - (D) **Nombrar áreas:** cada palabra dentro de grid-template-areas define una región (por ejemplo, header, sidebar, main).
   - (E) **Posicionar items:** los elementos se asignan mediante la propiedad grid-area, asociándolos directamente por nombre.
3. **Columnas Flexibles (C):** muestra la capacidad de crear rejillas dinámicas con repeat(auto-fit, minmax()).
   - (F) La combinación repeat(auto-fit, minmax(200px, 1fr)) genera automáticamente tantas columnas como el espacio permita, asegurando una maquetación fluida y responsiva.

Este esquema resume las dos grandes revoluciones del Grid avanzado:

- **Estructura semántica (áreas nombradas)**
- **Estructura adaptable (columnas automáticas y proporcionales)**



# Caso de Estudio: El layout de Spotify

## Contexto

La interfaz de Spotify Desktop y Web Player es un ejemplo paradigmático de layout modular, adaptable y semántico. En la pantalla principal encontramos tres zonas visualmente diferenciadas:

- Una sidebar fija con el menú de navegación.
- Un main-content central con listas de reproducción y álbumes.
- Una now-playing-bar en la parte inferior.

Esta disposición se podría construir de manera clara con `grid-template-areas`.

## Estrategia

Definir un contenedor principal con áreas nombradas:

```
.container {
  display: grid;
  grid-template-columns: 250px 1fr;
  grid-template-rows: auto 100px;
  grid-template-areas:
    "sidebar main"
    "nowplaying nowplaying";}

.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.nowplaying { grid-area: nowplaying; }
```

De esta forma:

- La sidebar ocupa la primera columna.
- El contenido principal llena el resto.
- La barra de reproducción (now-playing-bar) se extiende por debajo.

Cuando el diseño necesita adaptarse a pantallas pequeñas, basta con modificar la estructura textual:

```
@media (max-width: 768px) {
  .container {
    grid-template-columns: 1fr;
    grid-template-areas:
      "sidebar"
      "main"
      "nowplaying";
  }
}
```

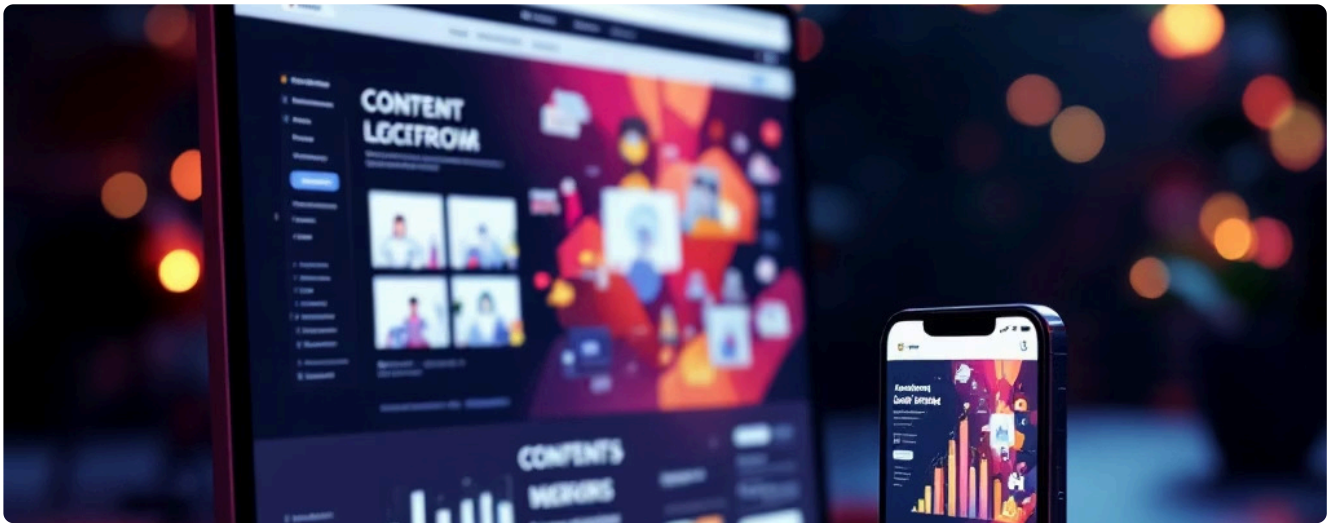
El layout se convierte automáticamente en una estructura vertical, optimizada para móviles.

## Resultado

- El código es legible y declarativo: se entiende a simple vista qué representa cada zona.
- No se requiere JavaScript para reorganizar el contenido.
- La experiencia del usuario es coherente en todos los dispositivos.
- El mantenimiento es mínimo: un único cambio en `grid-template-areas` redefine toda la estructura.

## Impacto real

El enfoque declarativo de `grid-template-areas` ha inspirado a múltiples empresas tecnológicas, como Spotify, Medium, Figma y Notion, a adoptar estructuras similares para interfaces responsivas. El resultado son aplicaciones visualmente consistentes, con CSS más limpio y menos dependencias externas.



# Herramientas y Consejos para tu Futuro Profesional

## Piensa en "áreas", no en coordenadas

Cuando diseñes con Grid, imagina zonas de contenido: "cabecera", "menú", "contenido", "pie". Usa nombres claros y consistentes. Esto hará que tu CSS sea autoexplicativo, como si fuera un mapa visual de la interfaz.

## La sintaxis de grid-template-areas es un dibujo

Cada línea de texto representa una fila, y cada palabra, una celda o un bloque. Ejemplo:

```
grid-template-areas:
  "header header"
  "sidebar main"
  "footer footer";
```

Es como un boceto dentro del CSS: intuitivo y legible incluso para quien no programó el layout.

## Usa minmax() para evitar layouts rotos

Define límites mínimos y máximos para tus columnas:

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

Así te aseguras de que las tarjetas o imágenes nunca se reduzcan demasiado ni se salgan de la cuadrícula.

## Comprende la diferencia entre auto-fit y auto-fill

- **auto-fit** colapsa columnas vacías, ajustando el tamaño de las existentes.
- **auto-fill** reserva espacio para columnas que podrían llenarse después.

En la práctica, auto-fit es ideal para galerías responsivas que se expanden de forma natural.

## Herramientas recomendadas:

- **Grid Garden** → juego interactivo para aprender los fundamentos.
- **CSS Grid Generator (by Sarah Drasner)** → herramienta visual para generar código de Grid.
- **Firefox DevTools** → Grid Inspector → muestra líneas, áreas y tamaños de la cuadrícula con colores.
- **Figma o Penpot** → útiles para planificar layouts que luego traducirás a Grid.

## Combina Grid + Flexbox

Grid estructura la página; Flexbox alinea el contenido dentro de cada celda. Por ejemplo:

- Usa Grid para crear la disposición general (cabecera, contenido, pie).
- Usa Flexbox dentro del header para alinear el logo y el menú.

# Mitos y Realidades

### ❌ Mito: "grid-template-areas es solo para prototipos, no para producción."

→ **FALSO.** Al contrario, `grid-template-areas` es una de las herramientas más potentes para entornos de producción. Su sintaxis legible hace que el código sea auto-documentado y fácil de modificar en equipos grandes. Grandes empresas lo usan en interfaces reales, precisamente por su claridad y mantenibilidad.

### ❌ Mito: "Para hacer una galería responsiva necesito muchas media queries."

→ **FALSO.** Gracias a repeat(auto-fit, minmax(250px, 1fr)), puedes crear una galería totalmente adaptativa sin escribir una sola media query. El navegador calcula automáticamente cuántas columnas caben y ajusta el tamaño. Esta combinación reemplaza docenas de líneas de CSS tradicional, simplificando radicalmente la gestión del diseño.

---

## Resumen Final

- `grid-template-areas` → permite nombrar zonas de la cuadrícula y asignar elementos por nombre.
- `grid-area` → posiciona cada elemento dentro de su área correspondiente.
- `minmax()` → define rangos flexibles de tamaño para filas o columnas.
- `auto-fit` y `auto-fill` → crean layouts fluidos que se adaptan automáticamente al ancho del contenedor.
- Puedes diseñar layouts responsivos sin media queries, reduciendo el código y aumentando la legibilidad.
- Grid avanzado combina semántica, flexibilidad y potencia: el trinomio perfecto para el diseño web moderno.