Aspectos adicionales sobre CSS

CSS Grid y Flexbox

Luis Valencia Cabrera (lvalencia@us.es)

Research Group on Natural Computing
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

13-02-2026, Bases de Datos

Índice

Panorámica

- Hay muchos aspectos básicos en CSS relacionados con la organización de la página (su *layout*), algunos de los cuales hemos visto:
  - Modelo de cajas, revisando aspectos como el **margin**, **padding**, dimensiones **width** y **height** o cómo afecta a este modelo el valor de la propiedad **box-sizing**.
  - Visualización, con propiedades como **display** (con valores `block`, `inline`, `inline-block` o `none`, entre otros) y **visibility** (analizando la diferencia entre `display: none` y `visibility: hidden`).
  - Posicionamiento, con la propiedad **position**, pasando del valor por defecto `static` a algún tipo de posicionamiento especial (`relative`, `absolute`, `fixed` o `sticky`)

Limitaciones y soluciones

- Las variantes de posicionamiento y visualización anteriores permiten trabajar bien con elementos individuales...
  - pero se quedan cortas para organizar globalmente las webs, con necesidades complejas y flexibles.
- Ahí surgen dos tecnologías dentro de CSS que se han establecido como las tendencias que dominan el mercado:
  - **Grid** (*CSS Grid Layout*), para dos dimensiones, disponiendo los elementos por filas y columnas formando una cuadrícula.
  - **Flex** (Flexbox, *CSS Flexbox Layout*), para disponer elementos en una dimensión, ayudando a disponer y alinear los elementos horizontal o verticalmente.
  - Ambos se pueden combinar, con organización general basada en *grid*, y algunos bloques dentro usando *flex*.

Flex vs Grid: ¿cuál usar?

¿Qué es más recomendable usar?

- Depende. Como regla general...
  - ¿**Solo** necesito organizar elementos por filas **o** por columnas? Mejor **flexbox**.
  - ¿**Necesito** organizar por filas **y** columnas? Mejor **grid**.

Índice

Grid

CSS Grid permite:

- Marcar un contenedor como grid, con `display: grid`
- Establecer una serie de columnas dentro del contenedor, por ejemplo con `grid-template-columns`
- Indicar cierta separación entre elementos, con `gap` (mismo espacio entre filas y columnas) o bien `column-gap` y `row-gap` (para dar distinto espacio por cada dimensión)

```
.mi_container {
   display: grid;
   grid-template-columns: 1fr 1fr 1fr;
   gap: 2rem;
}
```

Se puede ver la idea general aquí, aunque se aconseja ver las propiedades interactivamente en Grid garden.

Establecer un grid

Para indicar que un contenedor se debe comportar como un grid, basta con asignarle:

```
.mi_container {
   display: grid;
}
```

Hay una variante de línea (para ocupar por defecto el espacio necesario para alojar los elementos contenidos, en lugar de todo el ancho del espacio en el que se encuentre el contenedor):

```
.mi_container {
   display: inline-grid;
}
```

Grid: propiedades esenciales

grid-template-columns

La propiedad `grid-template-columns` permite:

- Distribuir los elementos en columnas, estableciendo sus anchuras.
- Asignar una combinación de anchos fijos (absolutos px, relativos rem, etc.), porcentuales (%), fracciones del espacio sobrante (fr)...

```
.mi_container {
   display: grid;
   grid-template-columns: 50px 1fr 2fr 1fr 3rem;
}
```

Por ejemplo, en este grid dejamos una primera columna de 50 píxeles, una última de 3 veces el ancho de la letra m, y el espacio restante se divide en 3 partes, con tres columnas tales que la central ocupa el doble que las dos que tiene a izquierda y derecha. Podemos ver la idea general y más detalles aquí.

Grid: propiedades esenciales

grid-template-rows

La propiedad `grid-template-rows` se comporta de forma similar a `grid-template-columns`, pero aplicado a las filas, y como tal permite:

- Distribuir los elementos en filas, estableciendo sus alturas.
- Asignar una combinación de alturas fijas (absolutos px, relativos rem, etc.), porcentuales (%), fracciones del espacio sobrante (fr)...

```
.mi_container {
   display: grid;
   grid-template-columns: 50px 1fr 2fr 1fr 3rem;
   grid-template-rows: 20% 50% 30%;
}
```

Grid: propiedades esenciales

grid-template

Podemos emplear el atajo `grid-template` para proporcionar tanto las filas propias de `grid-template-rows` como las columnas de `grid-template-columns`, en ese orden.

```
.mi_container {
   display: grid;
   grid-template: 100px 1fr / 2fr 3fr 2fr;
}
```

Así, en este ejemplo se establece un grid con dos filas (la primera con 100 píxeles y la segunda con el resto de la altura del contenedor) y con 3 columnas (la primera y la tercera con 2/7 del espacio horizontal disponible cada una, y la segunda con 3/7 de dicho espacio). Hay más información en esta dirección y ejemplos interactivos aquí.

Grid: propiedades esenciales

Espacio entre filas y/o columnas

- La propiedad `gap` establece una separación, un espacio, entre cada fila y columna.
- Si queremos indicar distintas separaciones para filas y columnas, emplearemos en su lugar:
  - `column-gap` para el espacio entre columnas
  - `row-gap` para el espacio entre filas
- También se puede utilizar `gap` como atajo, pasándole ambos el espacio para filas y para columnas: `gap: 2rem 1rem`, como vemos aquí.

Grid: propiedades esenciales

Tamaño implícito versus explícito

- Como se detalla en esta dirección, si usamos las propiedades anteriores de tipo `grid-template` estamos estableciendo las dimensiones de las filas o columnas explícitamente.
- Las dimensiones no establecidas se ajustarán al tamaño de su contenido.
- No obstante, se puede indicar un tamaño por defecto para los elementos no explicitados anteriormente, con propiedades como `grid-auto-rows` o `grid-auto-columns`.

```
.mi_container {
   display: grid;
   grid-template-columns: 1fr 2fr 2fr;
   grid-auto-rows: 100px;
}
```

Grid: propiedades esenciales

Valores especiales

Además de valores dados en unidades convencionales, tenemos funciones para generar otros valores:

- `repeat`: permite repetir un valor determinado el número de veces indicado. Por ejemplo, para indicar que queremos 4 columnas de 2fr cada una, escribiríamos `grid-template-columns: repeat(4,2fr);`
- `minmax`: como se explica aquí, permite crear tamaños variables dentro de un rango.
- `auto-fill`: permite que el número de columnas dependa del tamaño de la ventana, rellenando con tantas columnas como quepan.

Grid: propiedades sobre los hijos del contenedor

Posiciones de elementos en el grid

Tenemos propiedades con las que podemos indicar las posiciones de inicio y/o fin de los elementos dentro del grid. Las posiciones no se refieren exactamente a las celdas del grid sino a las líneas de separación entre los elementos, comenzando con los límites exteriores del grid.

- `grid-column-start`: indica la línea vertical en la que empieza el elemento.
- `grid-column-end`: indica la línea vertical en la que termina (si no se indica, es `grid-column-start` + 1, ocupando así una columna del grid).
- `grid-row-start`: indica la línea horizontal en la que empieza el elemento.
- `grid-row-end`: indica la línea horizontal en la que termina (si no se indica, es `grid-row-start` + 1, ocupando así una fila del grid).

Grid: propiedades sobre los hijos del contenedor

Posiciones de elementos en el grid - Consideraciones

- Podemos indicar un start mayor que el end, y funciona. También podemos indicar valores negativos para cualquiera de las propiedades, comenzando a contar por el final.
- En lugar de los valores de ambas líneas de inicio y fin, podemos indicar uno de los valores, y en el otro un `span` (indicando el número de columnas - o filas, en su caso - que ocupa el elemento):

```
.mi_elemento {
   grid-column-end: 5;
   grid-column-start: span 3;
}
```

Grid: propiedades sobre los hijos del contenedor

Atajos

- Podemos compactar con propiedades abreviadas, con instrucciones como `grid-column: 2 / 5` o `grid-row: 3 / 6`.
- Aún podemos abreviar más, con algo como `grid-area: 1 / 2 / 4 / 6`, que indica que empecemos en la línea que delimita la primera fila, segunda columna, y terminemos con la línea correspondiente a la cuarta fila, sexta columna.

Grid: propiedades sobre los hijos del contenedor

Otras propiedades - order

- Podemos alterar el orden relativo de un elemento dentro de su fila, pasando a una columna anterior o posterior dependiendo de su `order` (un mayor valor desplazará a la derecha, y menor a la izquierda).
- Por defecto, todo elemento tiene order 0, de modo que un valor negativo desplaza al elemento a la izquierda y uno positivo a la derecha.

Posicionamiento con areas

Como se explica en esta dirección, podemos establecer la estructura general con `grid-template-areas` y luego indicar en los elementos a posicionar la propiedad `grid-area`.
Podemos ver más ejemplos aquí.

Posicionamiento con areas - Ejemplo

```
.container {
   display: grid;
   grid-template-columns: 50px 50px 50px 50px;
   grid-template-rows: auto;
   grid-template-areas:
      "header header header header"
      "main main . sidebar"
      "footer footer footer footer";
}
.item-a {
   grid-area: header;
}
.item-b {
   grid-area: main;
}
.item-c {
   grid-area: sidebar;
}
.item-d {
   grid-area: footer;
}
```

Índice

Flexbox

Flexbox permite:

- Marcar un contenedor como flexbox, con `display: flex`
- Establecer una dirección, con `flex-direction`
- Organizar los elementos en la dirección elegida, con `justify-content`

```
.mi_flexbox {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
}
```

Vamos a analizar muchas más propiedades interactivamente a través de Flexbox froggy

Flexbox: propiedades esenciales

flex-direction

Esta propiedad determina si el contenedor organizará los elementos por filas, columnas, o sus correspondientes variantes invertidas. Puede adoptar los siguientes valores:

- `row` (valor por defecto, organizar los elementos en filas)
- `column` (por columnas)
- `row-reverse` (por filas, en orden inverso)
- `column-reverse` (por columnas, en orden inverso)

```
.mi_flexbox {
  display: flex;
  flex-direction: column;
}
```

Flexbox: propiedades esenciales

justify-content

Alinea los elementos en función del `flex-direction`:

- `flex-start`: se alinean los elementos al principio del contenedor (izquierda si es `row`, derecha si es `row-reverse`, arriba si es `column`, abajo si es `column-reverse`).
- `flex-end`: se alinean al final.
- `center`: centra en la dimensión elegida (horizontal o vertical).
- `space-between`: deja el máximo espacio posible entre los elementos, con los extremos pegados a los límites del contenedor.
- `space-around`: deja el mismo espacio a izquierda y derecha (si es `row`) o arriba y abajo (si es `column`) de cada hijo del contenedor.
- `space-evenly`: hace que el espacio entre cada dos elementos hijos del contenedor sea siempre el mismo (incluyendo el espacio hasta el inicio y el final del contenedor).

Flexbox: propiedades esenciales

align-items

Alinea los elementos en la dirección complementaria a la de `flex-direction` (verticalmente si tenemos `row`, horizontalmente si tenemos `column`):

- `flex-start`: alinea al inicio.
- `flex-end`: al final.
- `center`: centrado.
- `baseline`: alinea a la base de los elementos.
- `stretch`: los elementos ocupan por completo la dimensión complementaria a la indicada por `flex-direction`.

Flexbox: otras propiedades

Para hijos:

- `align-self`: cambia el alineamiento en dirección complementaria del elemento indicado.
- `order`: cambia el orden relativo de un elemento dentro de su contenedor, lo que condicionará su posición.

Para el contenedor:

- `flex-wrap`: pasa los elementos que no quepan a la fila (respec. columna) siguiente, en lugar de forzar todos en la misma fila (respec. columna).
- `flex-flow`: dota de un atajo para agrupar a flex-direction y flex-wrap.
- `align-content`: indica cómo distribuir filas (respec. columnas) en la dirección complementaria. Solo tiene sentido si se ha establecido flex-wrap.

Propiedades "de flexbox" en grid

Podemos usar en CSS grid las propiedades `justify-content` y `align-content` vistas en flexbox:

- `justify-content` alineará horizontalmente.
- `align-content` lo hará verticalmente.

En ambos casos tenemos los mismos valores posibles: `start` (al inicio), `end` (al final), `center` (centrado), `stretch` (ocupando todo el espacio), y `space-around`, `space-between`, `space-evenly`, con el mismo criterio que en el caso de flex.