Introducción (y un poco de historia)

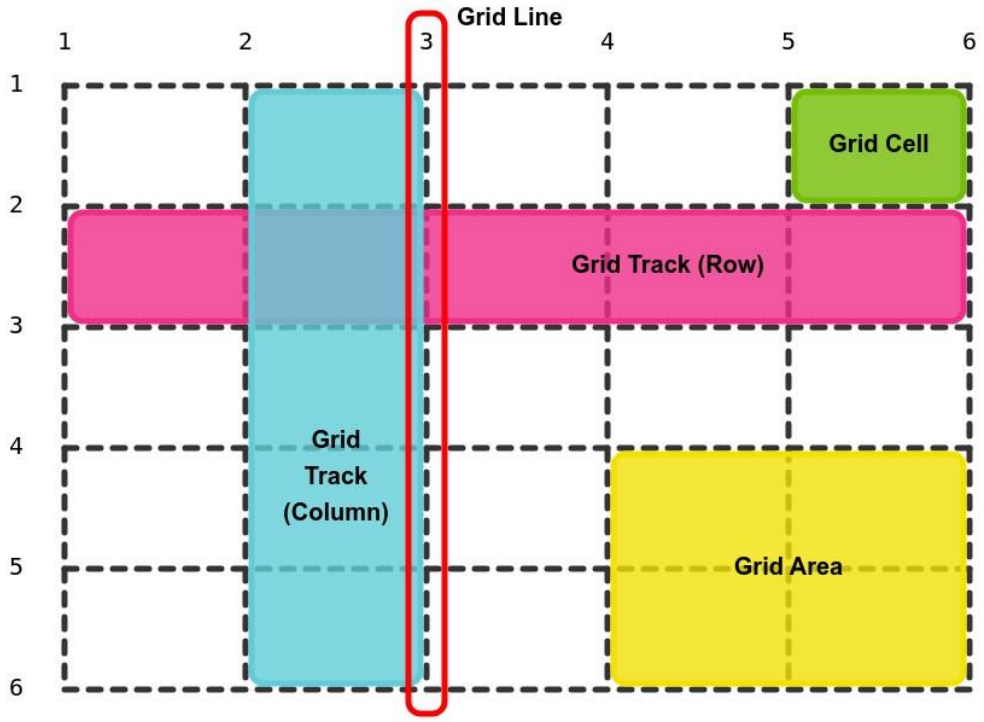
El diseño complejo de las webs ha ido evolucionando a lo largo de su historia. Desde el primer uso de tablas como elemento de diseño hasta posteriormente los frames o el modelo de cajas y float… hasta flex y ahora grid, la última pieza en el juego.

La primera referencia que existe a este módulo de diseño es de 2011 y se implementó sobre Microsoft Explorer. En 2017 los principales navegadores (Chrome, Firefox, Safari y Edge) soportaban ya CSS grid sin prefijos. Sin embargo aún hoy CSS Grid no está reconocido como estándard por la W3C y sólo se le reconoce como un "Candidato Recomendado".

Tampoco tiene por el momento mucha aceptación entre los frameworks CSS mas populares. Sólo lo utiliza Tailwind. Bootstrap no utiliza grid e implementa su propio sistema de rejillas mediante el uso de Flex. Foundation tampoco se decide a incorporarlo.

Grid comparte muchas similitudes con Flex y al igual que él nos permite realizar diseños complejos de forma fácil mientras que antes para hacer las mismas cosas teníamos que recurrir a trucos mas o menos complejos. Pero tal vez la diferencia más importante entre ambas es que Flex nos permite diseños en una única dimensión mientras que Grid permite también diseños bidimensionales.

El modelo de diseño que nos propone grid es el siguiente:



Propiedades

Al igual que nos ocurría con flex, los diseños con grid constan de dos partes. Un contenedor o elemento padre dentro del cual tenemos un número de elementos hijos o items. Asimismo, también tenemos un conjunto de propiedades bastante extensas que nos permiten definir nuestro diseño. Las principales (aunque veremos alguna otra) son las siguientes:

Propiedad	Explicación
`display`	grid o inline-grid. Si usamos la segunda propiedad nuestro grid se comporta como un elemento en línea y no de bloque.
`grid-template-columns`	Especifica el número de columnas y el tamaño relativo de cada una de ellas
`grid-template-rows`	Especifica el alto de cada fila
`column-gap`	Espacio de separación entre columnas. Puedes verlo por ahí como grid-column-gap. Es lo mismo, pero la forma preferida ahora es esta
`row-gap`	Espacio de separación entre filas. Puedes verlo por ahí como grid-row-gap. Es lo mismo, pero la forma preferida ahora es esta
`gap`	Un "atajo" para los dos anteriores. También puedes verlo como grid-gap, pero está desaconsejado
`grid-column-start`	Especifica la "linea-grid" donde empieza el item
`grid-column-end`	Especifica la "línea-grid" donde termina el item
`grid-column`	Atajo para los dos anteriores
`grid-row-start`	Especifica la "linea-grid" donde empieza el item
`grid-row-end`	Especifica la "línea-grid" donde termina el item
`grid-row`	Atajo para los dos anteriores
`grid-template-areas`	Permite definir áreas en el contenedor del grid
`grid-area`	Permite asignar un ítem a un area previamente definida
`grid-auto-columns`	Especifica el tamaño por defecto de las columnas cuyo tamaño no haya sido definido
`grid-auto-rows`	Especifica el tamaño por defecto de las filas
`grid-auto-flow`	
`grid`	Atajo para grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns y grid-auto-flow

Nuestros primeros diseños

Partiremos del siguiente diseño html para ir introduciendo las diferentes propiedades:

```
<div class="contenedor">
      <div class="caja">En un lugar de la Mancha de cuyo nombre no quiero
            acordarme no ha mucho que vivía un hidalgo caballero•••</div>
      <div class="caja">1</div>
      <div class="caja">2</div>
      <div class="caja">3</div>
      <div class="caja">4</div>
      <div class="caja">S</div>
      <div class="caja">6</div>
      <div class="caja">7</div>
      <div class="caja">8</div>
</div>
```

Y vamos a probar con este primer CSS:

```
body {
  background: #e6e6e6;
      }

.contenedor {
  display: grid;
  grid-template-columns: 30% 38% 30%;
  gap: 2rem 1%;
  }

.caja {
  background: #f1e9f7;
  border: 3px solid #a579cc;
  color: #a579cc;
  padding: 2rem;
  font-size: 2rem;
  text-align: center;
}
```

Las propiedades aplicadas al body y a los items no tienen nada que ver con grid. Son meramente estéticas. El resultado que deberías de ver es este:

Fíjate que aparte de activar grid con la propiedad display solo usamos dos propiedades. Con grid template columns definimos el número de columnas (tres, porque son los elementos que aparecen como valor) y su proporción frente al 100% del espacio del navegador. Hemos calculado que sumen un 98% y nos sobra un 2% que usaremos en la segunda propiedad como espaciado entre columnas (un 1% entre cada una de ellas). Por último, grid-gap es un atajo que especifica el espaciado entre filas en primer lugar y entre columnas en segundo equivalente a usar grid-row-gap y grid-column-gap)

También podemos usar la unidad de medida fr:

```
grid-template-columns: 1fr 3fr 2fr;
```

fr es una unidad que se define como una fracción de un todo. En el ejemplo anterior suponemos el total del espacio disponible dividido en 6 fracciones iguales y asignamos una a la primera columna, 3 a la central y 2 a la última. Esta unidad tiene en cuenta el espacio que asignemos a los gaps de forma que no tenemos que hacer cálculos complejos.

La unidad fr también puede aplicarse a las filas:

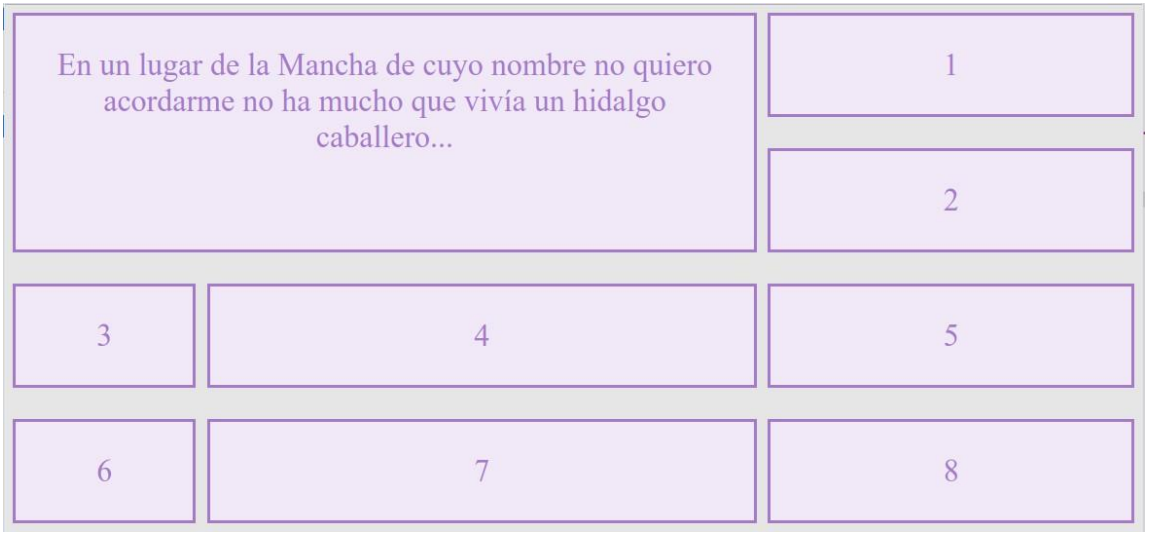
```
grid-template-rows: 3fr 2fr 1fr;
```

En este caso lo que nos indica es que la primera linea sera el triple de alta que la primera y la segunda el doble que la tercera.

Vamos ahora a tocar los elementos del grid. Las propiedades grid-column-start, grid-column-end, grid-row-start y grid-rown-end nos permiten modificar la ubicación y el espacio que ocupan. Por ejemplo así:

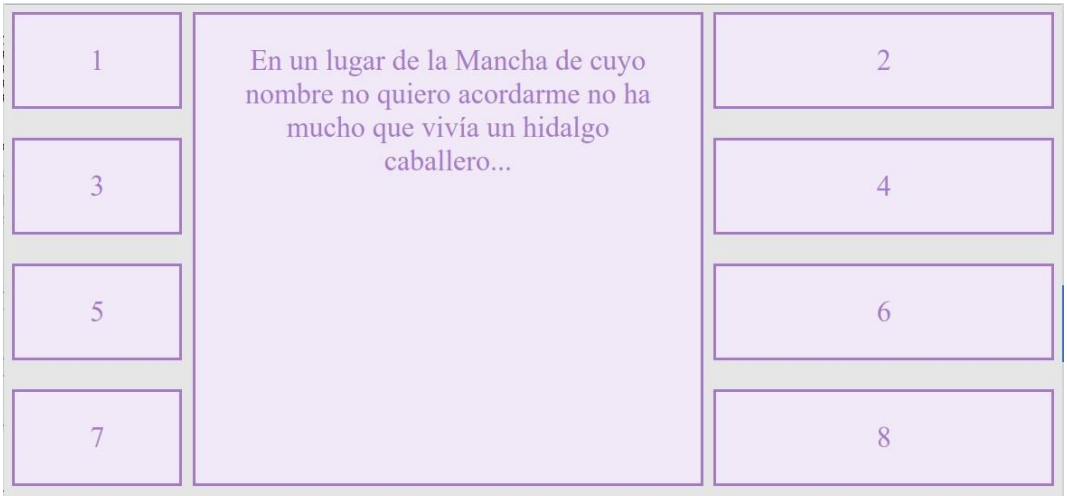
```
.caja:first-child{
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

Obteniendo este resultado:



O así:

```
.caja:first-child{
  grid-column-start: 2;
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 5;
}
```



La sintaxis que se usa con esta propiedad es un poco engañosa: Los números que se usan no se refieren a las filas o a las columnas sino a las "lineas grid" que vimos en el modelo de diseño inicial de este documento. Admite valores negativos siendo -1 la última línea, -2 la penúltima, etc.

grid-column es un atajo para grid-column-start y grid-column-end. Grid-row igual para las filas. Además, tenemos disponible otra sintaxis, tal vez un poco mas clara, así:

```
grid-column: 2 / span 2;
grid-row: 1 / span 2;
```

El primer número indica la posición de inicio. Span seguido de un número indica el número de items que ocupará a partir del sitio donde empieza el item.

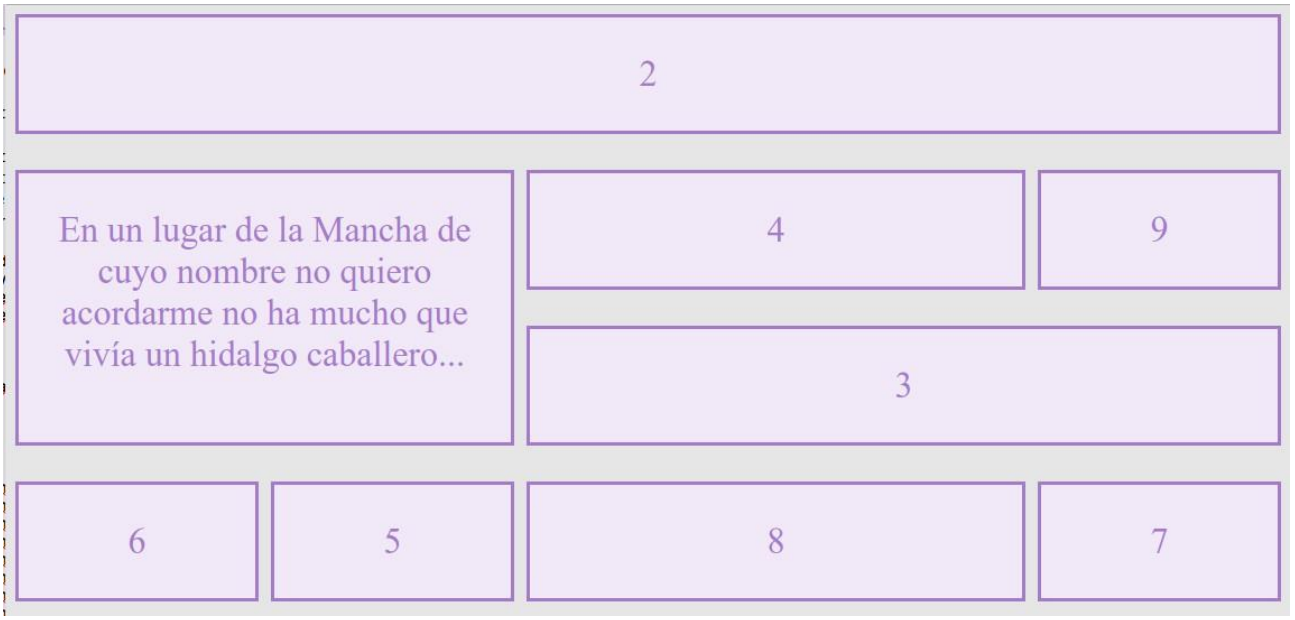
Las propiedades grid-template-area (aplicadas al contenedor) y grid-area (a los items) nos permite hacer una especie de mapa de nuestro grid y asignar los diferentes items en el lugar en el que queremos que aparezca. Veamos un ejemplo. Cambiemos la definición de nuestro contenedor por esta:

```
.contenedor {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr;
  grid-template-areas:
          "area1 area1 area1 area1 area1"
          "area2 area2 area3 area3 area3"
          "area2 area2 area4 area4 area4";
  gap: 2rem 1%;
  }
```

Y ahora los items:

```
.caja:nth-child(l){   grid-area: area2; }
.caja:nth-child(2){   grid-area: areal; }
.caja:nth-child(3){   grid-area: area4; }
.caja:nth-child(4){   grid-area: area3; }
.caja:nth-child(S){   grid-area: area6; }
.caja:nth-child(6){   grid-area: areaS; }
.caja:nth-child(7){   grid-area: area8; }
.caja:nth-child(8){   grid-area: area7; }
.caja:nth-child(9){   grid-area: area9; }
```

El resultado es este:



Si queremos dejar un espacio en blanco sin asignar a ningún área ponemos un punto en su lugar. Prueba a cambiar el mapa del anterior template por este:

```
grid-template-areas:
         "areal areal areal areal areal"
         "area2 area2 area3 area3 area9"
         "area2 area2 . area4 area4"
         "areaS area6 area7 area7 area8";
```

Podemos también delimitar el área que ocupa cualquier item de nuestro grid con la siguiente sintaxis:

```
grid-area: 2 / l / 4 / 3;
```

Donde los cuatro números hacen referencia a las lineas-grid que delimitan el area que ocupa el item en el siguiente orden: fila de inicio / columna de inicio / fila de fin / columna de fin

Por defecto los items ocupan todo el espacio disponible, pero podemos cambiar esto con las propiedades justify-items, align-item (en el contenedor) y justify-self y aling-self en los items.

Los valores que pueden tomar estas propiedades son start, end, center y stretch. Son autoexplicativas. Juega un poco con ellas y lo veras.

Existen tres propiedades mas de las que no hemos hablado: grid-auto-rows y grid-auto-columns nos permiten definir el tamaño de filas y columnas que no hayan sido definidas de ninguna otra forma. Por último, grid-auto-flow nos permite modificar la forma en que se organizan nuestros items, emplazándolos ordenados por filas (por defecto) o por columnas.

Una segunda forma de manejar nuestro diseño es poniendo nombre a las grid-lines, esas líneas imaginarias que delimitan nuestra refilla. Lo hacemos así:

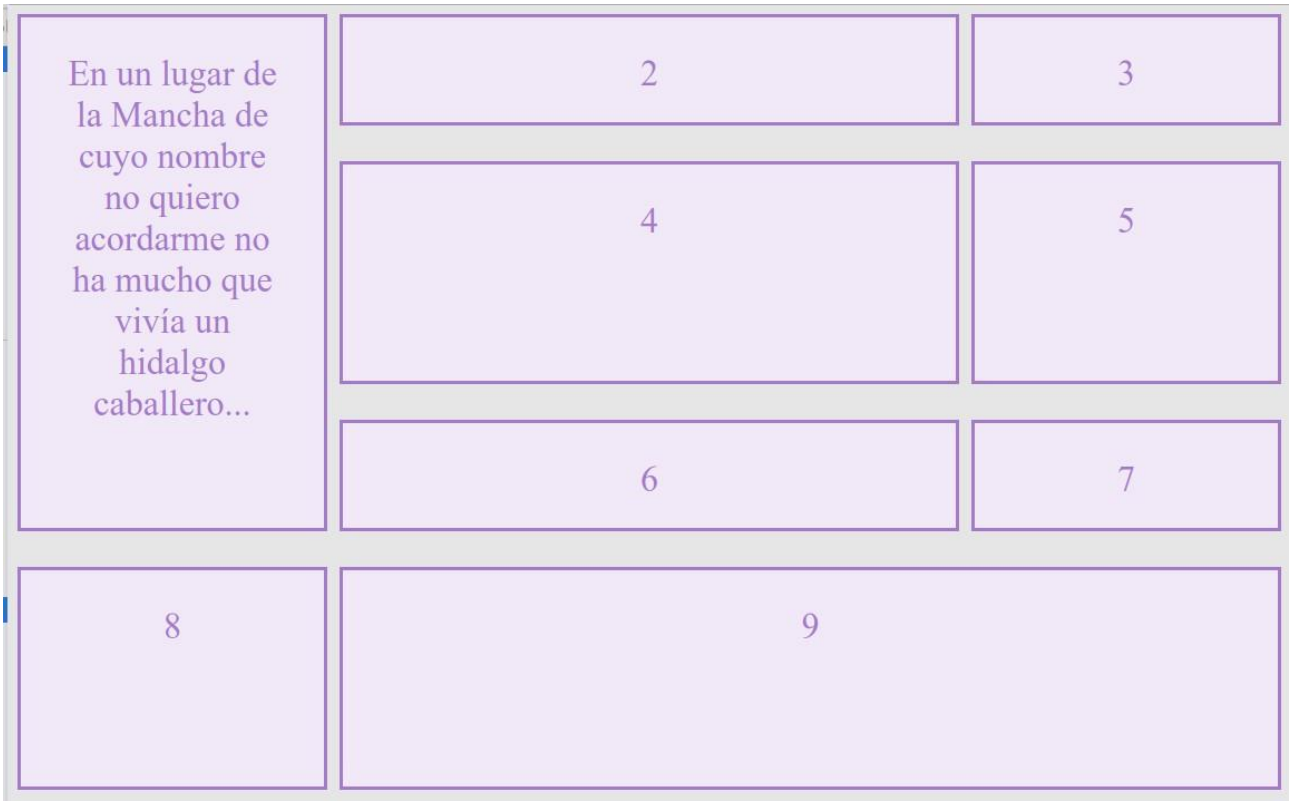
```
.contenedor {
  display: grid;
  grid-template-columns: [uno-c] lfr [dos-c] 2fr [tres-c] lfr [cuatro-c];
  grid-template-rows: [uno-f] l00px [dos-f] 200px [tres-f] l00px
           [cuatro-f] 200px [cinco-f];
  gap: 2rem l%;
  }
```

A continuación podemos definir los límites de nuestros items usando las propiedades que ya hemos visto pero usando los nombres de las líneas, lo cual lo hace mucho mas intuitivo:

```
.caja:nth-child(l){
     grid-row-start: uno-f;
     grid-row-end: cuatro-f;
}

.caja:nth-child(9){
     grid-column-start: dos-c;
     grid-column-end: cuatro-c;
}
```

El resultado sería este:



Diseños responsivos

Podemos dejar en manos del navegador la forma en que se ubican los items usando las funciónes repeat() y minmax(). Prueba con esta definición del contenedor:

```
.contenedor {
  display: grid;
  width: 50%;
  grid-template-rows: repeat(auto-fit, 100px);
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 2rem 1%;
  }
```

repeat admite un montón de opciones interesantes y es muy flexible. Por ejemplo, la siguiente definición nos crearía un grid de tres columnas y cuatro filas. La primera columna de l00px, las otras dos iguales hasta llenar el ancho del navegador. Las filas serían de dos tamaños diferentes de forma que la segunda y la cuarta son del doble de alto que la primera y la segunda

```
grid-template-columns: 100px repeat(2, 1fr) ;
grid-template-rows: repeat(2, 1fr 2fr);
```

autofit y autofill son dos valores que indican al grid o bien ajuste el tamaño a lo que le indicamos (autofit) o que expanda las celdas hasta llenar el espacio disponible (autofill).

Pero, tal vez, la mejor aproximación que podemos hacer para un diseño responsivo es combinar medias querys con la distribución por áreas. Por ejemplo, podríamos definir así nuestro contenedor grid:

```
.contenedor {
  display: grid;
  grid-template-areas:
           "areal"
           "area2"
           "area3"
           "area4"
           "areaS"
           "area6"
           "area7"
           "area8"
           "area9";
  gap: 2rem l%;
  align-items: center;
  }
```

```
@media (min-width:768px) {
      .contenedor {
            display: grid;
            grid-template-areas:
                  "areal areal"
                  "area2 area3"
                  "area4 areaS"
                  "area6 area7"
                  "area8 area9";
      gap: 2rem l%;
      align-items: center;
      }
}

@media (min-width:l024px) {
      .contenedor {
            display: grid;
            grid-template-areas:
                  "areal areal areal"
                  "area2 area3 area4"
                  "areaS area6 area4"
                  "area7 area8 area4"
                  "area9 area9 area9";
      gap: 2rem l%;
      align-items: center;
      }
}
```

De esta forma la disposición de las diferentes áreas del grid y el espacio que ocupan cambiará

en función del ancho de la pantalla de nuestro navegador. En el ejemplo anterior establecemos tres "breakpoints" que se disparan cuando la pantalla tiene un ancho superior a 768px, a 1024px o cuando es inferior a 768px (valor por defecto) pero podemos establecer tantos como queramos con muy poco esfuerzo.