silocreativo.com

Introducción y primeros pasos con CSS Grid (II) • SiloCreativo

Alfonso Serrano

10-13 minutos

Seguimos en esta serie de textos que nos sirven de **introducción a CSS Grid**. En el artículo anterior, vimos como CSS Grid nos permitía crear de forma muy sencilla una rejilla en nuestro ***Grid Container***. Y es justo en ese punto donde continuaremos.

Como ya comprobasteis, aunque la rejilla era un gran paso para nuestra maquetación, resultaba bastante insatisfactoria si queríamos realizar un diseño completo. Por ello, lo próximo que debemos conocer es **como posicionar y mover todos los diferentes elementos que componen nuestra web dentro de la rejilla**. Es decir, nos meteremos de lleno en la manipulación de los *Grid Items*.

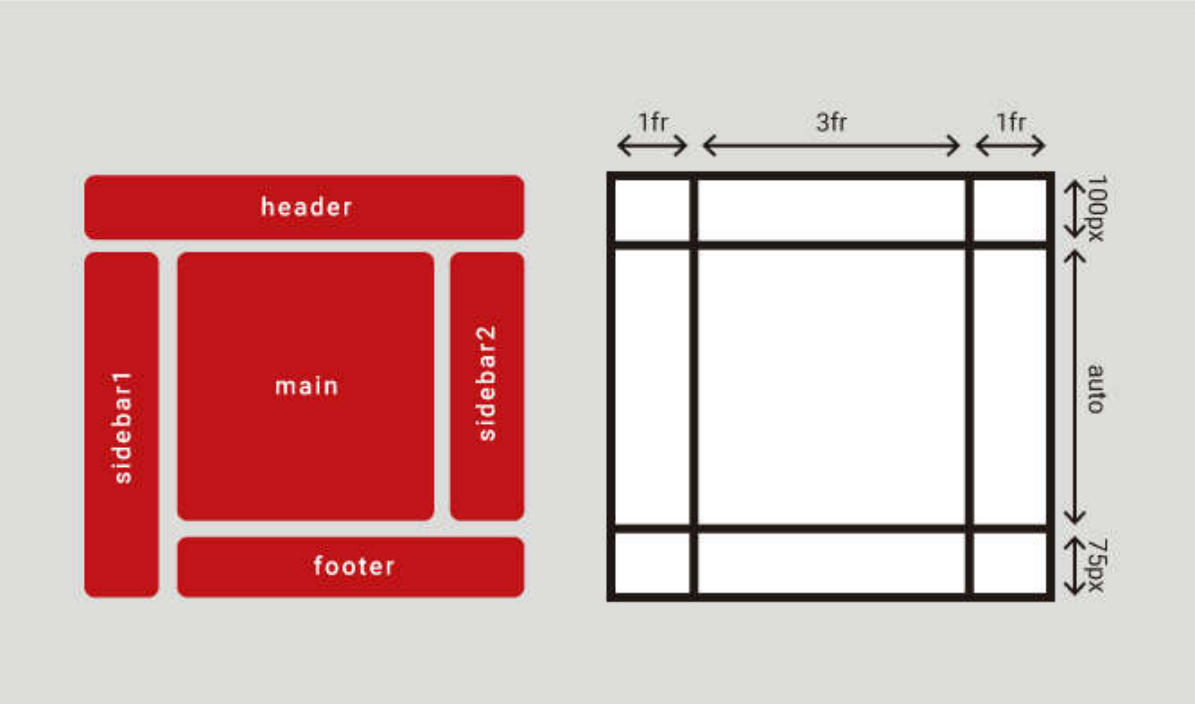
Como siempre, antes de continuar, y si sois recien iniciados en CSS, os recomendamos que echéis un vistazo a nuestro artículo, que seguro que os resolverá más de una duda que os pueda surgir en este texto, ya que habrá que escribir bastante código.

Posicionando elementos: Grid Items

Mediante diferentes propiedades aplicadas a los *Grid Items* seremos capaces de posicionar nuestros elementos dentro de la

rejilla. Pero antes que nada, hay que crear dicha rejilla.

Tomaremos un diseño básico, pero con los suficientes contenedores como para que no nos resulte demasiado aburrido, que es lo último que pretendemos. Nuestro ejemplo se compone de un *header*, dos *sidebars*, un cuerpo y un *footer*. Para que resulte más sencillo de entender la posición y dimensiones de cada uno de los elementos, una pequeña imagen de muestra:



Como ya vimos, es **extremadamente sencillo de construir esta maquetación gracias a CSS Grid**. En el artículo anterior hicimos una rejilla parecida, por lo que no os resultará muy difícil de construirla por vosotros mismos. En cualquier caso, el HTML que se usará será este:

```
<div class="container">
        <div class="header"> </div>
        <div class="sidebar1"> </div>
        <div class="main"> </div>
        <div class="sidebar2"> </div>
```

```
        <div class="footer"> </div>
</div>
```

Si habéis visto este código y os resulta algo confuso, quizás sería el momento idóneo para refrescar un poco vuestros conocimientos sobre HTML.

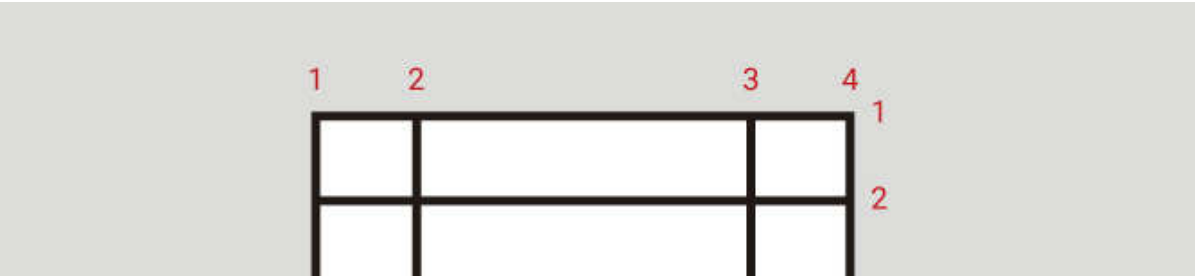
Y en CSS, simplemente hay que escribir una simple rejilla de 3×3.

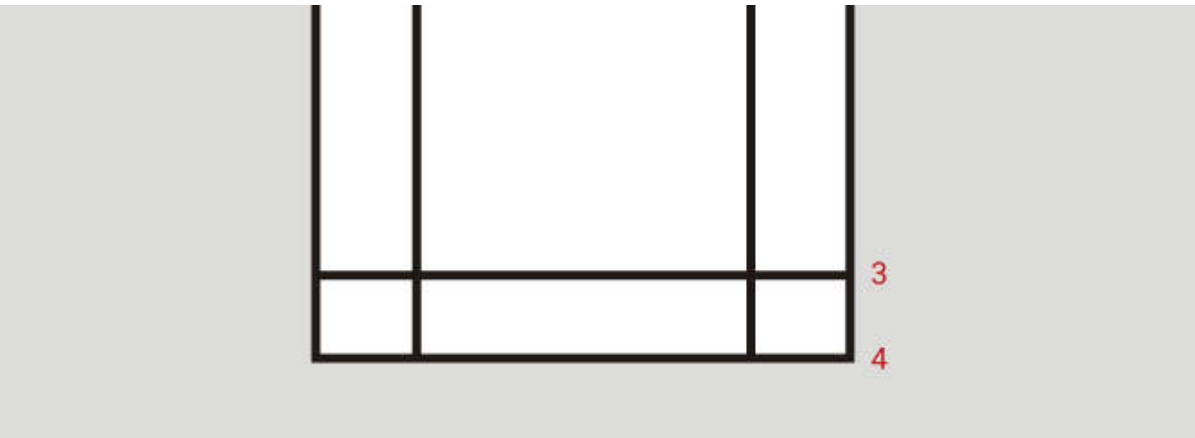
```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
}
```

Pero ahora llega el momento de la verdad. **¿Cómo posicionamos cada uno de los *divs* existentes dentro de nuestro *container* en la rejilla que hemos creado?** Vamos a ver como hacerlo mediante diferentes métodos, para que podáis elegir por vosotros mismo cual es el que os resulta más adecuado según vuestra forma de trabajar.

Posicionamento mediante Grid Lines

Como sabemos, llamamos *Grid Lines* a cada una de las líneas que componen nuestra rejilla. **Por defecto, CSS Grid ordena numéricamente esta rejilla** en orden ascendente, de izquierda a derecha las columnas, o líneas verticales, y de arriba hacia abajo las filas, o líneas horizontales.





Con todas estas piezas, vamos a empezar a ver las primeras propiedades que vamos a usar, que en este caso serán cuatro. ***Grid-row-start*** **y** ***grid-row-end*** le dirán a nuestro *Grid Item* en que fila empieza y acaba, respectivamente. Y ***grid-colunm-start*** **y** ***grid column-end*** tendrán la misma función, pero con respecto a las columnas. Como ejemplo, vamos a posicionar nuestro *header*, teniendo en la cabeza tanto la imagen con el diseño que tratamos de poner en pie, como la **posición y numeración de cada una de las líneas**.

```
.header {
        grid-row-start: 1;
        grid-row-end: 2;
        grid-column-start: 1;
        grid-column-end: 4;
}
```

Muy sencillo, como veis. Solo es tener claro la **posición que ocupa dentro de nuestra rejilla, y marcarle sus límites**. Como extra, vamos a ver como podemos hacer exactamente lo mismo, pero **ahorrando un par de líneas**. Y además nos servirá de introducción para el valor *span*.

```
.header {
        grid-row: 1;
```

```
        grid-column: 1 / 4;
}
```

Mediante ***grid-row*** **y** ***grid-column*** **podemos indicar en una sola línea tanto el inicio como el final, separados ambos por una barra inclinada**. Puede parecer un ahorro mínimo, pero en un archivo CSS completo, es una cantidad de tiempo extra que estamos ganando. Y en SiloCreativo somos muy ahorradores de tiempo en nuestro trabajo.

Y no nos quedemos solo en el *header*, posicionemos en resto de nuestros elementos:

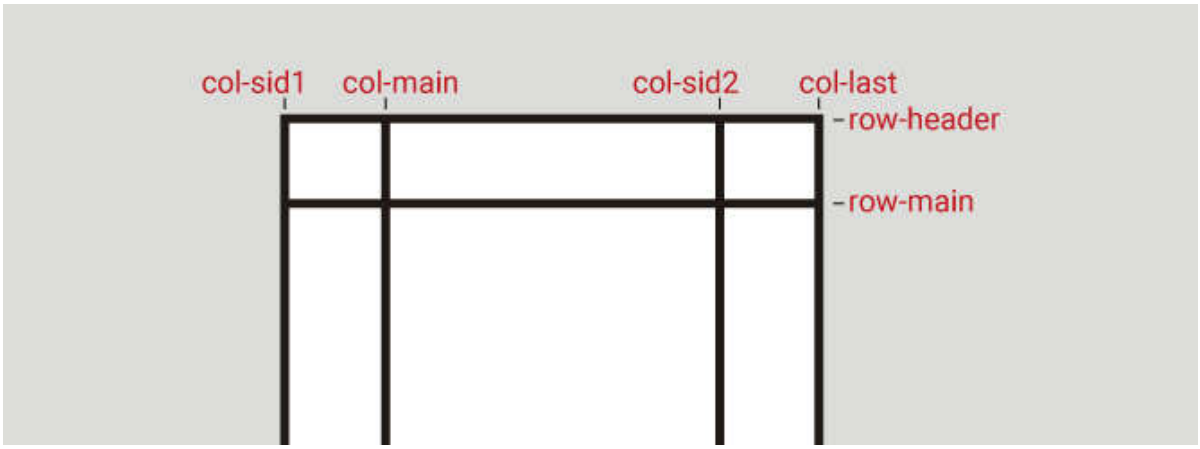
```
.header {
        grid-row: 1;
        grid-column: 1 / 4;
}

.sidebar1 {
        grid-row: 2 / 4;
        grid-column: 1;
}

.main {
        grid-row: 2;
        grid-column: 2;
}

.sidebar2 {
        grid-row: 2;
        grid-column: 3;
}
```

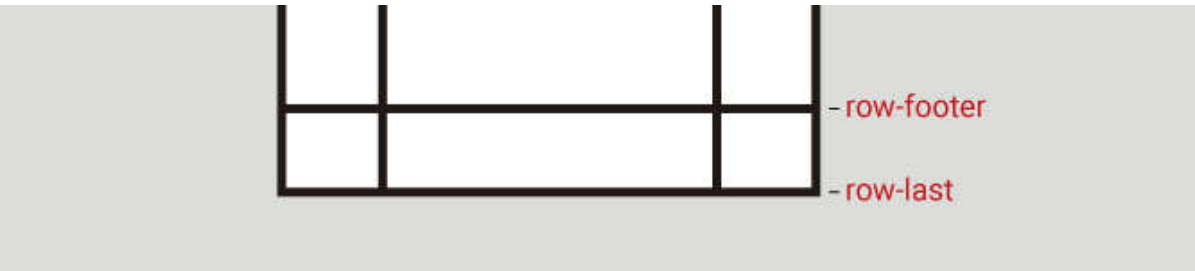
```
.footer {
        grid-row: 3;
        grid-column: 2 / 4;
}

.header {
        grid-row: 1;
        grid-column: 1 / span 3;
}
```

Posicionamento mediante Grid Lines (explicit)

Muchos podréis decir: *«esto está muy bien, pero si mi rejilla es muy densa, puedo llegar a perderme entre tanto número, y no saber hasta que fila o columna en concreto llegan mis elementos»*. Pues CSS Grid también ofrece una solución para vosotros, ya que podemos darle **un nombre a cada linea, ya sea fila o columna**.

Sigamos con el mismo diseño que estamos usando hasta ahora, pero voy a nombrar cada línea a mi gusto. Vosotros podéis ponerle el nombre que queríais, aunque lo ideal es que sea lo **suficientemente significativo** como para que, solo con ver ese nombre escrito, sepamos que posición ocupa dentro de nuestra rejilla. Estos son los que yo he elegido.





Una vez los tengamos, nos vamos de nuevo a nuestro *Grid Container*, y modificaremos nuestros valores de *grid-template-rows* y *grid-template-columns*. Me gusta mucho esta opción porque **la escritura es realmente gráfica**, y ahora comprenderéis porque digo esto.

Si vamos viendo nuestra rejilla de arriba hacia abajo (en el orden que siguen los *rows*), nos encontramos primero con nuestra primera línea horizontal, luego el primer *grid track*, luego la segunda línea horizontal, etc. Y así hasta llegar hasta abajo.

Pues en nuestro CSS, **lo escribiremos siguiendo este mismo orden**: primero el nombre que le hayamos dado a nuestra primera línea horizontal (entre corchetes), a continuación el tamaño de la primera fila, luego el nombre de de la segunda línea horizontal (entre corchetes), etc. Y exactamente lo mismo para las columnas, solo que el orden será de izquierda a derecha.

Según los nombres que le he dado a mis líneas de rejilla, así quedaría mi CSS:

```
.container{
        display: grid;
        grid-template-rows: [row-header] 100px
[row-main] auto [row-footer] 75px [row-last];
        grid-template-columns: [col-sid1] 1fr
[col-main] 3fr [col-sid2] 1fr [col-last];
}
```

Y a continuación, **asignaría a mis Grid Items los nombres que he elegido**:

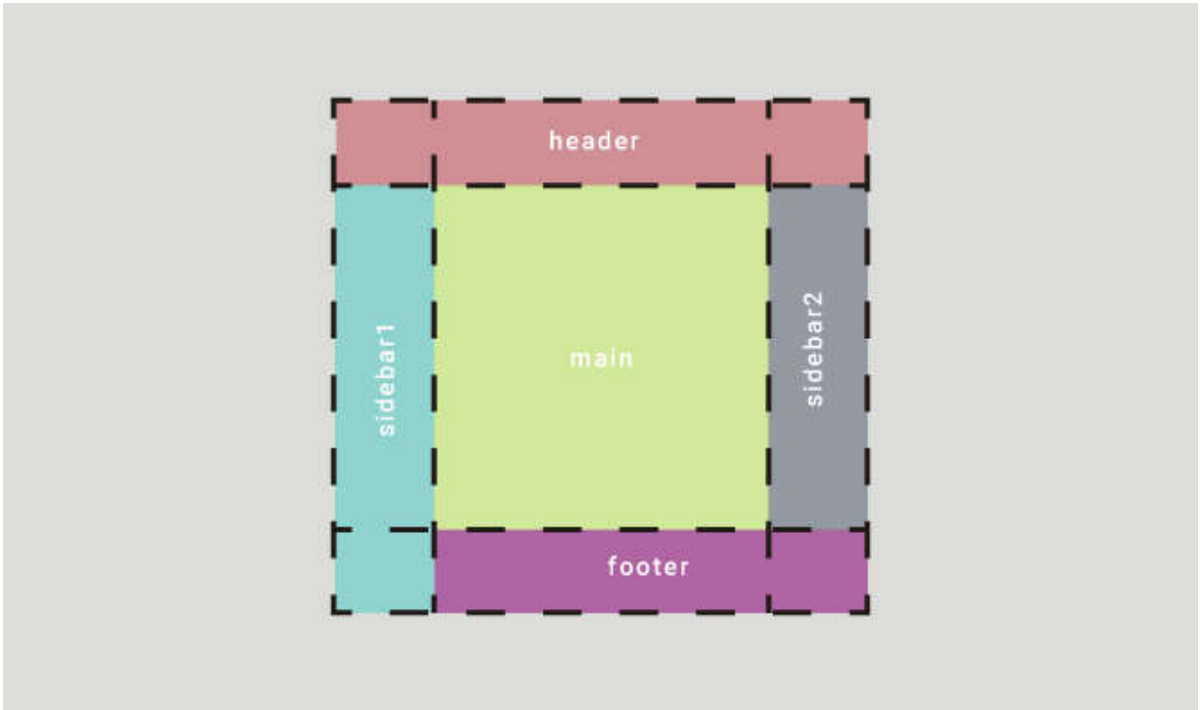
```
.header {
        grid-row: row-header;
        grid-column: col-sid1 / col-last;
}

.sidebar1 {
        grid-row: row-main / row-last;
        grid-column: col-sid1;
}

.main {
        grid-row: row-main;
        grid-column: col-main;
}

.sidebar2 {
        grid-row: row-main;
        grid-column: col-sid2;
}

.footer {
        grid-row: row-footer;
        grid-column: col-main / col-last;
}
```

Fantástico! **Ahora nuestro código es mucho más personal**, y nuestros *Grid Items* no están definidos solo por unos fríos y desangelados números.

Posicionamiento mediante Grid Areas

Otro de los elementos que definían nuestra rejilla eran las *Grid Areas*. Quizás pudisteis pensar que era algo secundario, ya que no hemos visto nada de ellas. Pero **vamos a comprobar que es tremendamente potente** y, bajo mi punto de vista, mucho más sencillo de usar que el método anterior para posicionar nuestros elementos.

En primer lugar, refresquemos nuestra memoria con el diseño que nos ha acompañado hasta ahora, **superponiendo en esta ocasión cada uno de los elementos en la rejilla**.



Como se puede ver claramente, tenemos elementos que ocupan solo un espacio, mientras que otros ocupan varios. No pasa nada. **Mediante la propiedad *grid-area* aplicada a nuestros *Grid Items*, asignaremos los límites tanto verticales como horizontales de nuestros elementos**. Es como si cogiésemos las propiedades *grid-row* y *grid-column*, y las comprimiésemos en una sola. En el caso de las *grid-area*, el orden que seguiríamos sería

grid-area: límite superior / limite izquierdo / límite inferior / límite derecho. Veamos como quedaría en este caso nuestro CSS.

```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
}

.header {
        grid-area: 1 / 1 / 2 / 4;
}

.sidebar1 {
        grid-area: 2 / 1 / 4 / 2;
}

.main {
        grid-area: 2 / 2 / 3 / 3;
}

.sidebar2 {
        grid-area: 2 / 3 / 3 / 4;
}

.footer {
        grid-area: 3 / 2 / 4 / 4;
}
```

En este caso estamos usando los números que nos adjudica por defecto *CSS Grid* para las *Grid Lines*, pero perfectamente

podemos asignarle nombres y usarlos como propiedades de las *Grid Area*. Las limitaciones las imponemos nosotros según nos sea más sencillo de trabajar.

Y por último, **explico mi método favorito**. Si ya dijimos que era muy gráfico de escribir las propiedades de las grid-template-rows y grid-template-columns porque seguíamos un orden visual (de arriba a abajo, y de izquierda a derecha), **con la propiedad *grid-template-areas* esto se magnifica**.

En primer lugar, asignaremos a cada uno de nuestros *Grid Items* la propiedad *grid-area*, y su valor será el nombre que queramos. Una vez tengamos esto. volveremos a nuestro *Grid Container*, y escribiremos la propiedad *grid-template-areas*. En nuestro caso, como nuestra rejilla es de 3×3, el valor del *grid-template-areas* tendrá tres valores en una primera fila, tres valores en la segunda, y tres valores en la tercera, por lo que gráficamente dará la impresión de que tengamos una rejilla de palabra de 3×3. **Esto no será casual, ya que vamos a posicionar cada uno de los valores que dimos a los *grid-area* en la posición que ocupa dentro de esta *rejilla de valores CSS***. Sin miedo, directamente. Veamos cual sería el resultado.

```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
        grid-template-areas: "head head head"
                             "side1 main side2"
                             "side1 foot foot"
}
```

```
.header {
        grid-area: head;
}

.sidebar1 {
        grid-area: side1;
}

.main {
        grid-area: main;
}

.sidebar2 {
        grid-area: side2;
}

.footer {
        grid-area: foot;
}
```

Prácticamente, **hemos dibujado nuestro rejilla y sus elementos dentro de nuestro código CSS**. No solo es muy intuitivo, y sencillo de hacer, sino que si en un futuro los elementos modifican posición, aparecen otros nuevos o incluso desaparecen, los cambios serán tremendamente fáciles de realizar, ya que **de un simple vistazo, podemos imaginarnos como queda nuestra maquetación**.

Pero, **¿qué ocurre si en nuestra rejilla, hay un área que es un espacio vacío y no contiene ningún elemento?** Pues muy fácil, solo hay que poner un punto en la posición donde iría el valor

correspondiente, y CSS Grid entenderá que en ese espacio de rejilla no le corresponde que vaya ningún elemento.

Y hasta aquí todo lo que quería enseñaros sobre posicionamiento en CSS Grid. No me quería despedir sin dejaros **algunos enlaces de interés**, como la web de Jen Simmons, que nos muestra muchos de los experimentos que hace con CSS Grid, los cuales podéis replicar para practicar. O el canal de Youtube de Rachel Andrew, que tiene videos cortos sobre cada uno de las propiedades de CSS Grid.

Para el próximo y último artículo de esta serie sobre CSS Grid, veremos como **superponer elementos y ordenarlos**, ademas de aprender algunas propiedades para que **nuestros diseños con CSS Grid sean responsive**. ¡Espero que os guste!