CSS Grid Layout: Haciéndolo Responsivo

by [Ian Yates](https://tutsplus.com/authors/ian-yates)3 Oct 2016

Difficulty:BeginnerLength:ShortLanguages:

[CSS Grid Layout](https://webdesign.tutsplus.com/es/categories/css-grid-layout)[CSS](https://webdesign.tutsplus.com/es/categories/css)[Responsive Web Design](https://webdesign.tutsplus.com/es/categories/responsive-web-design)

This post is part of a series called [Understanding the CSS Grid Layout Module](https://webdesign.tutsplus.com/series/understanding-the-css-grid-layout-module--cms-1079).

[CSS Grid Layout: Using Grid Areas](https://webdesign.tutsplus.com/tutorials/css-grid-layout-using-grid-areas--cms-27264)

[Understanding the CSS Grid “Auto-Placement Algorithm”](https://webdesign.tutsplus.com/tutorials/understanding-the-css-grid-auto-placement-algorithm--cms-27563)

Spanish (Español) translation by [Javier Salesi](https://tutsplus.com/authors/javier-salesi) (you can also [view the original English article](https://webdesign.tutsplus.com/tutorials/css-grid-layout-going-responsive--cms-27270?ec_unit=translation-info-language))

A través de ésta serie nos hemos familiarizado con la sintaxis de Grid, aprendido sobre algunas formas eficientes de maquetar elementos en una página, y nos hemos despedido de viejos hábitos. En éste tutorial vamos a aplicar todo eso en un diseño web responsivo práctico.

**Media Queries**

Utilicemos el demo desde donde nos quedamos la última vez.

Comprende dos grillas declaradas; nuestra grilla principal y la grilla anidada dentro de uno de nuestros elementos. Podemos controlar cuándo éstas grillas tienen efecto usando media queries, lo que significa que podemos completamente redefinir nuestro maquetado en anchuras de diferentes ventanas gráficas.

Comenzamos por duplicar la primera declaración de grid y encerrando el duplicado en una media-query que tiene como prioridad los dispositivos móviles (estoy usando 500px como el punto de quiebre, pero eso es completamente arbitrario):

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | .grid-1 {      /\* grid declaration styles \*/  }      @media only screen and (min-width: 500px) {        .grid-1 {          /\* grid declaration styles \*/      }    } |

Ahora, dentro de la primera declaración cambiaremos como es definida nuestra grilla, colocando todo en una sola columna. Enlistaremos sólo una columna en nuestra regla grid-template-columns, asegurándonos que las cuatro filas que ahora tenemos sean definidas con grid-template-rows, y arreglamos el maquetado con grid-template-areas :

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | .grid-1 {      display: grid;      width: 100%;      margin: 0 auto;        grid-template-columns: 1fr;      grid-template-rows: 80px auto auto 80px;      grid-gap: 10px;        grid-template-areas:    "header"                              "main"                              "sidebar"                              "footer";  } |

También hemos hecho nuestro grid-gap de sólo 10px por defecto, para tomar en cuenta las pantallas más pequeñas.

[Aquí está lo que nos da.](http://codepen.io/tutsplus/pen/NRbOGL/) Notarás que también estamos usando nuestra media query para cambiar el padding y font-size en nuestros elementos de .grid-1 div .

**Nuestra Grilla Anidada**

Eso se hace cargo del maquetado principal, ,pero aún tenemos la grilla anidada que continua apegándose a su maquetado de dos columnas bajo toda circunstancia. Para arreglar ello haremos exactamente lo mismo que antes, pero usando un punto de quiebre diferente para sugerir un planteamiento primero-contenido.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | .item-2 {      /\* grid declaration styles \*/  }      @media only screen and (min-width: 600px) {        .item-2 {          /\* grid declaration styles \*/      }    } |

Revisa el [resultado final en CodePen](http://codepen.io/tutsplus/pen/mAOzVq/).

Un par de cosas a notar aquí:

* Como lo dijimos antes, puedes arreglar *visualmente* los elementos de la grilla sin importar el orden original, y las media queries significan que tenemos diferentes órdenes visuales para diferentes anchuras de pantalla. Sin embargo, al anidar se tiene que seguir respetando el origen; nuestros elementos de la grilla anidada deben siempre ser visualmente y *realmente* descendientes de su padre.
* Transiciones CSS no tienen ninguna influencia sobre Grid Layout. Cuando nuestras media queries hacen su parte, y nuestras áreas de grilla se mueven a sus nuevas posiciones, no podrás hacerlas que se acomoden en su lugar.

**auto-fill y minmax()**

Otro planteamiento responsivo en Grid es adecuado a maquetados de tipo mosaico; bloques que se dimensionan y fluyen automáticamente, dependiendo de la ventana gráfica.

**auto-fill**

Nuestro planteamiento hasta ahora ha sido determinar cuántos trazos hay y cuidar que los elementos encajen cosecuentemente. Eso es lo que ocurre en éste demo; tenemos grid-template-columns: repeat(4, 1fr) ; que dice "crea cuatro columnas, y haz cada una con una anchura de una unidad de fracción".

Con la palabra reservada auto-fill podemos dictar que tan *anchos* son nuestros trazos y dejamos que Grid descubra cuántos encajarán en el espacio disponible. En éste demo hemos usado grid-template-columns:repeat(auto-fill, 9em); que indica "haz las columnas con una anchura de 9em cada una, y encaja tantas como puedas en el contenedor grid".

**Nota**: ésto también toma en cuenta nuestros medianiles, el grid-gap.

El contenedor en éstos demos tiene un fondo oscuro para mostrar claramente cuánto espacio hay disponible, y verás que no ha sido llenado completamente en el último ejemplo. ¿Así que cómo hacemos eso?

**minmax()**

La función minmax() nos permite establecer un tamaño mínimo y un máximo para un trazo, permitiendo a Grid que trabaje en ellos. Por ejemplo podríamos configurar tres columnas, las primeras dos de una anchura de 1fr, la última de una anchura de 1fr máximo, pero no reduciéndola menos de 160px:

|  |  |
| --- | --- |
| 1 | grid-template-columns: 1fr 1fr minmax(160px, 1fr); |

Todas las columnas se reducirán cuando achiques la ventana, pero la última columna sólo será empujada hasta ese ancho. [Observa](http://codepen.io/tutsplus/pen/rrjywv/).

De regreso a nuestro demo auto-fill, si fuéramos a cambiar nuestro ancho de columna por minmax(9em, 1fr) Grid colocaría tantos trazos de 9 em como fuera posible, pero entonces los expandería a un máximo de 1fr hasta que el contenedor estuviera lleno:

**Advertencia**: Grid recalculará los trazos al recargar la página (trata de encoger la ventana del navegador y presiona actualizar) pero no lo hará así al redimensionarse la ventana. Las media queries pueden ser usadas para alterar los valores, pero no funcionarán bien al redimensionarse la ventana

**Conclusión**

Terminemos con algunos puntos conceptuales:

* Las media queries pueden ayudarnos a reacomodar *completamente* los maquetados con Grid al redefinir grid-template-areas (y otras cosas) para diferentes escenarios.
* Transiciones CSS no tienen ningún efecto en los cambios realizados en el maquetado grid.
* La palabra reservada auto-fill es útil para llenar los contenedores grid.
* La función minmax() complementa perfectamente a auto-fill, asegurando que los contenedores sean llenados adecuadamente, pero no hace "responsivo" el maquetado en el verdadero sentido de la palabra.

Con las lecciones aprendidas en ésta serie, ¡estás preparado para salir y comenzar a utilizar Grid! Manténte atento a más tutoriales sobre Grid, ejercicios prácticos, soluciones a problemas comunes de maquetación y actualizaciones.

**Recursos Útiles**

* [Ejemplo 29 de Grid: minmax() y expandiendo columnas y filas](http://codepen.io/rachelandrew/pen/GZQYQa) de Rachel Andrew.
* Video: Rachel Andrew (obviamente) [mostrando minmax() en el maquetado de la página de inicio de Tuts+](https://www.youtube.com/watch?v=paBou8Fy25k&feature=youtu.be)
* Borrador del Editor de W3C: [auto-fill](https://drafts.csswg.org/css-grid/#valdef-repeat-auto-fill)