silocreativo.com

Ejemplo de Landing Page con CSS Grid: Apollo 8 (II) • Silo Creativo

Alfonso Serrano

8-10 minutos

Si hace unas semanas estuvimos creando el diseño de nuestra landing page en Affinity Designer, hoy damos el siguiente paso. Y no es más que **crear de forma definitiva nuestra página sobre Apollo 8**. Para ello, daremos uso de algunas de las propiedades que nos ofrece CSS Grid.

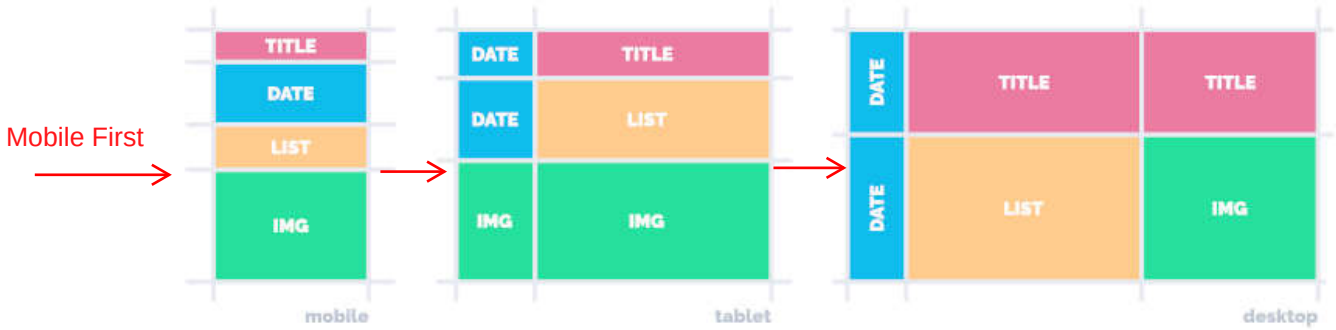
En primer lugar, os recomendamos que leáis nuestros dos artículos de introducción a CSS Grid (aquí tenéis el primer artículo y el segundo artículo). Por otras parte, también podéis leer nuestro texto sobre CSS, para los que estáis empezando con este lenguaje de programación.

Como vimos en la anterior parte, vamos a ir poco a poco, **teniendo en consideración los aspectos y propiedades de CSS Grid más características**. Viendo su funcionamiento, y porqué los aplicamos en cada caso.

Sección Hero, con diseño fluido usando grid-areas

En esta parte de nuestra web, tal como vimos en el anterior artículo, vimos como cada uno de sus elementos variaban de posición según el ancho que tuviéramos en nuestro navegador.

Teníamos ciertos puntos de ruptura, donde pasábamos de un diseño a tres columnas a la máxima anchura, a uno de una sola columna para pantallas más pequeñas. **Nos ayudaremos de los Media Queries para crear estos puntos de ruptura.** Aunque antes de todo, si no estas muy puesto al día, te recomendamos que leas antes nuestro artículos sobre los Media Queries para ver su funcionamiento.



Para esta parte del diseño, vamos a dotar a cada elemento de un nombre específico. Esto nos facilitará mucho la tarea al usar `grid-template-areas`.

El código HTML que hemos escrito es bastante simple. Y como siempre deberíamos intentar hacer, **vamos a cuidar el orden de aparición de cada sección pensando en la accesibilidad**.

```
<section class="hero">
        <div class="hero-title"></div>
        <div class="hero-date"></div>
        <div class="names">
        <ul class="hero-list"></ul>
        </div>
        <div class="hero-img"></div>
</section>
```

Como ya hemos dicho, vamos a usar `grid-template-areas`

para crear la plantilla. Y para nombrar cada área, haremos uso de la función `grid-area`.

```
.hero-title {
        grid-area: title;
}

.hero-date {
        grid-area: date;
}

.hero-list {
        grid-area: list;
}

.hero-img {
        grid-area: img-hero;
}
```

Esto no nos creará aun ningún tipo de diseño. Pero antes de empezar con él, tenemos que aclarar otro asunto. En esta landing page, usaremos los principios del mobile first. Es decir, vamos a **empezar a diseñar para las pantallas más pequeñas y de smartphones**, y desde ahí, empezaremos a subir hasta llegar al diseño de escritorio. Y como el diseño en smartphone es muy sencillo, en una sola columna, su CSS correspondiente se organizará de la misma manera.

```
.hero {
        display: grid;
        grid-gap: 10px;
        grid-template-areas:
```

```
                "title"
                "date"
                "list"
                "img-hero";
}
```

Una vez tenemos esta CSS para smartphone, seguimos subiendo hasta un punto de ruptura intermedio de 500 px. En esta ocasión tenemos dos columnas, con un elemento que ocupa todo el ancho.

```
@media (min-width: 500px) {
        .hero {
                grid-template-columns: 1fr 4fr;
                grid-template-areas:
                        "date   title"
                        "date   list"
                        "img-hero       img-hero";
        }
}
```

A la primera columna le hemos dado un valor de 1fr, y a la segunda de 4fr. Es decir, que **la segunda columna siempre va a tener cuatro veces el ancho de la primera**. Además, como queremos que la imagen ocupe todo el ancho inferior, sus áreas ocupan esos dos espacios correspondientes.

Por último, vamos a crear el último punto de ruptura, en esta ocasión usando un diseño a tres columnas.

```
@media (min-width: 900px) {
        .hero {
                grid-template-columns: 1fr 4fr
3fr;
```

```
                grid-template-areas:
                        "date   title   title"
                        "date   list    img-hero";
        }
}
```

Seguimos usando la unidad de medida fr o fraction para asignar los valores de anchura a las columnas de nuestra plantilla. Si nos hemos fijado en el diseño inicial, veremos que el texto con **la fecha del lanzamiento del Apollo 8 aparece en vertical.** Para girar este texto, solo tendremos que añadir este código CSS:

```
.text-date {
    writing-mode: vertical-rl;
    transform: rotate(180deg);
}
```

Simplemente con esto ya tenemos lista nuestra primera sección, y totalmente funcional para cualquier dispositivo. **Personalmente, me resulta de mucha ayuda tener siempre a mano los bocetos o esquemas de mi diseño**, ya que me facilita mucho en el momento de escribir el código CSS con la posición de cada uno de los elementos. Que como habéis visto, son tremendamente gráficos.

Sección Galería, haciendo uso del auto posicionamiento

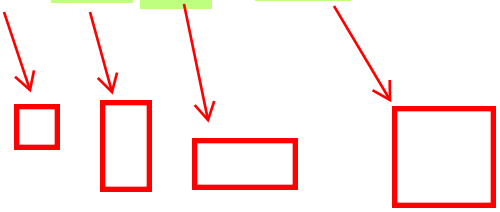
Para nuestra galería de imágenes no vamos a hacer uso de los Media Queries. Y esto se debe a que **queremos un diseño completamente fluido, y sin tener que describir ciertos puntos de ruptura** según el ancho de la web. Además de esto, vamos a tener imágenes a diferentes tamaños, que queremos que se auto-posicionen

breakpoint

posicionen, y que no dejen ningún espacio en blanco en nuestro diseño. Puede parecer que esto nos va a dar mucho trabajo. Y vamos a ver que es todo lo contrario. Como comentamos anteriormente, un gran ejemplo de este tipo de diseño lo tenemos en una de las demos de Jen Simmons, dentro de su Labs.

Empezaremos por asignar los tamaños de cada una de las fotografías. En nuestro diseño, tenemos 4 tipos de tamaños, en función de los *cells* que ocupen dentro de nuestra rejilla. Como dijimos en nuestro artículo de introducción a CSS Grid, un *cell* es la unidad mínima dentro de nuestra rejilla.

A cada imagen **le asignaremos una clase diferente en función de su tamaño**, según lo que ocupe: 1×1, 1×2, 2×1, o 2×2. Nos ayudaremos de span para denotarlos.



```
.img-small {
        grid-row: 1;
}

.img-med-vert {
        grid-row: 2;
}

.img-med-horz {
        grid-column: 2;
}

.img-big {
        grid-row: 2;
        grid-column: 2;
}
```

Seguir por aquí
01/02/2020 ,22:41:57

Según usemos `grid-row` o `grid-column`, le diremos a la imagen hacia que dirección (horizontal o vertical, respectivamente) se debe expandir y ocupar espacio. Y si usamos ambas, se deberá expandir en ambas direcciones.

Pues ya tenemos nuestra imágenes listas. **Pero nos queda lo más importante: crear nuestra rejilla.** Y para esto, usaremos simplemente tres propiedades de Grid: `auto-fit`, `minmax` y `grid-auto-flow`. Vamos a ver en primer lugar como quedaría nuestro CSS, y luego vamos comentando como funciona.

```
.gallery {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(175px, 1fr));
        grid-auto-rows: 175px;
        grid-auto-flow: dense;
}
```

Como siempre, lo primero en usar `display: grid`, para que el navegador sepa que vamos a usar Grid. A continuación, usamos las propiedad `repeat`, con el valor `auto-fit`. Con `auto-fit`, **se adaptan las columnas que haya disponibles, para que ocupen todo el ancho del contenedor**. Este valor lo combinamos con `minmax`, que le asigna a cada imagen un ancho mínimo y máximo en el que puede ajustarse.

En resumen, con estos dos valores conseguimos que **cuando una imagen o elemento alcance un ancho mínimo, automáticamente pase a la fila inferior**. Y al mismo tiempo, siempre estén ocupando el ancho total de su contenedor, para que no queden espacios en blanco.

Y ahora pasamos a mi propiedad favorita, `grid-auto-flow`. Si a esta propiedad le asignamos el valor `dense`, **todos los espacios en blanco que se van a ir creando a causa de tener elementos de diferentes tamaños se van a ir rellenando de manera automática**. Este es uno de los motivos por lo que tenemos imágenes que ocupan 1×1, ya que van a ser estas las que van a colocarse en estos huecos.

Por una parte, **vamos a tener siempre una galería perfecta, sin ningún espacio desaprovechado**. Pero por otra, si queremos que todo tenga un orden preciso, va a ser difícil de conseguir, ya que el navegador va a posicionar los elementos un poco a su antojo, según las necesidades que tenga. En este caso, como simplemente es una galería de imágenes en la cual, la cronología no tiene un peso importante, esta propiedad encaja a la perfección.

Conclusiones

Tal como hemos podido comprobar, la **resolución de problemas de posicionamiento de los elementos de una web se evitan de forma sencilla y eficaz con CSS Grid**. Nos ahorramos tanto tiempo de escritura, como de intentar averiguar como trasladar el diseño sobre papel o nuestro software de diseño vectorial favorito. Y esto se debe a que la traslación de uno a otro es casi directa.

Por otra parte, si hacemos uso de los Feature Queries que vimos en otro de nuestros artículos, **solucionaremos también las posibles dificultades que puedan existir con ciertos navegadores que aun no soportan CSS Grid**. Además de que no tendremos que volver a tocar código cuando lo soporten en el futuro.

Y esto es todo. Como siempre, tenéis disponible la sección de comentarios ante cualquier duda o sugerencia que nos queráis realizar.