silocreativo.com

Flexbox vs CSS Grid: Un ejemplo práctico • SiloCreativo

Alfonso Serrano

6-8 minutos

Para muchos de nosotros, cuando maquetábamos una web, era muy usual el uso de tablas, floats y otras propiedades que nos daba CSS, aunque resultaban unas herramientas algo ortopédicas si el diseño resultaba complejo.

Hace algunos año apareció **Flexbox**, que fue especialmente diseñado para crear pagina con un **diseño responsive**. Flexbox hizo que alinear elementos y su contenido sea mucho más sencillo, permitiéndonos crear páginas fluidas, flexibles y dinámicas, que funcionan en un amplio catálogo de dispositivos, simplemente usando CSS. Flexbox eliminó de una vez el desafió que suponía crear este tipo de páginas.

Pero desde hace algunos meses existe una nueva modulo herramienta, **CSS Grid**, que es **compatible de forma nativa con** Chrome 57 **y** Firefox 52, por lo que si tenéis vuestros navegadores actualizados, ya tendréis estas versiones, y esperemos que en un futuro próximo, sea compatible con el resto de navegadores.

Por esto, la pregunta es clara: ¿debo seguir usando Flexbox, o me conviene empezar a trabajar en mis proyectos con CSS Grid? Pues para sacarnos de dudas, vamos a crear una pequeña

plantilla, e intentaremos aplicar nuestro diseño usando tanto Flexbox como CSS Grid, para ver que nos resulta más adecuado.

Flexbox vs CSS Grid: nuestro HTML base

Como hemos dicho, **vamos a trabajar sobre un HTML básico**, sobre el que aplicaremos nuestro CSS, por una parte usando Flexbox, y por otra CSS Grid. Podéis trabajar directamente en CodePen, para poder seguir paso por paso todo lo que hacemos en el artículo.

Enlace a CodePen

Como veis, el diseño es muy simple, y ahora mismo un poco feo, pero queremos que sean bastante reconocibles cada una de sus secciones. Como vemos, nuestro HTML consiste básicamente en

un **header** con varios links, una **sección principal**, un **sidebar** y un **footer**, todo dentro de un contenedor. Con todo esto, intentaremos marcarnos varias metas, intentando mantener el la limpieza de nuestro código:

- Posición general de cada una de las cuatro secciones del HTML.
- Alinear los diferentes elementos del header.
- Hacer que la web tenga un diseño responsive.

Posicionamiento de las secciones

FLEXBOX

Empezaremos con la solución que usaríamos con Flexbox. Añadiremos un *display: flex* al contenedor, y posicionaremos a sus elementos hijos de forma vertical. Esto hará que las secciones se coloquen una debajo de otra.

```
.container {

display: flex;

flex-direction: column;

}
```

A continuación, necesitamos que la sección principal y el sidebar estén una junto a la otra. Como los **contenedores *flex* normalmente van en una dirección**, añadiremos un *wrapper* a nuestro HTML, que tendrá dentro tanto la sección principal como el sidebar. Y a este *wrapper* le añadiremos un *display: flex*, pero en esta ocasión en dirección horizontal.

aside

```
.wrapper {
    display: flex;
    flex-direction: row;
}
```

Solo nos queda por decidir las proporciones tanto de la sección principal como del sidebar. Por ejemplo, diremos que la sección principal tendrá **cuatro veces el tamaño del sidebar**, y al primero le añadiremos un margen para que haya cierto espacio entre cada sección.

```
.main {
    flex: 4;
    margin-right: 40px;
}
.sidebar {
   flex: 1;
}
```

CSS GRID

Existen varias maneras de usar CSS Grid, pero en esta ocasión nos ayudaremos de esta sintaxis. En primer lugar, definiremos **nuestras cuatro grid áreas**:

```
header {
    grid-area: header;
}
.main {
    grid-area: main;
}
.sidebar {
```

```
    grid-area: sidebar;
}
footer {
    grid-area: footer;
}
```

A continuación, podemos configurar la disposición de cada *grid-area*, con sus márgenes correspondientes:

```
.container {
    display: grid;
    grid-template-columns: 4fr 1fr;
    grid-template-areas:
        "header header"
        "main sidebar"
        "footer footer";
    grid-gap: 40px;
}
```

Aunque en un principio pueda parecer menos comprensible, una vez que sepamos usar la sintaxis de CSS Grid, vemos como su escritura resulta **incluso más sencilla** que en Flexbox.

Alinear elementos del header

FLEXBOX

Nuestro header tiene varios links, y por otro un botón, y queremos que los links estén justificados a la izquierda, y el botón a la derecha.

```
header {
    display: flex;
```

```
    justify-content: space-between;
}
```

Una vez tenemos tanto la lista de con los enlaces como el botón en su respectiva posición, haremos que todos los elementos de la lista de navegación se **alineen de forma horizontal**.

```
header nav {
    display: flex;
    align-items: baseline;
}
```

También se podía haber usado un *display: inline-block*, pero como queremos hacer Flexbox, vamos a usarlo para solucionar todo.

CSS GRID

Para separar la lista de navegación del botón, convertiremos el header en un *dispaly: grid* a dos columnas, y justificaremos cada una de ellas.

```
header{
    display: grid;
    grid-template-columns: 1fr 1fr;
}
header nav {
    justify-self: start;
}
header button {
    justify-self: end;
}
```

La alineación correcta de **los elementos de la lista de**

navegación no será tan precisa como queramos, ya que no disponemos de la opción *align-items* que si tenemos en Flexbox. Por lo tanto, tendremos que definir dentro otro grid.

```
header nav {
    display: grid;
    grid-template-columns: auto 1fr 1fr;
    align-items: end;
}
```

Aquí una de las **debilidades de CSS Grid**, ya que a pesar de la facilidad que nos da alinear contenedores, no es muy bueno para hacer lo mismo con los elementos que están dentro.

Hacer la web responsive

FLEXBOX

Por último, y no por ello menos importante, vamos a ver la solución que nos da Flexbox para cuando necesitamos hacer nuestra web responsive. Para ello, solo vamos a **cambiar la dirección del *wrapper***, de horizontal a vertical (o de *row* a *column*). Como es una página sencilla, solo usaremos un tamaño para nuestro *media-query*.

```
@media (max-width: 660px) {
    .main-and-sidebar-wrapper {
        flex-direction: column;
    }
```

CSS GRID

Como tenemos varias ***grid-areas*** definidas, las **reordenaremos**

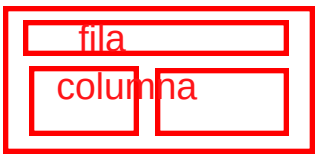
dentro de nuestro *media-query*.

```
@media (max-width: 660px) {
    .container {
        grid-template-areas:
            "header header"
            "main main"
            "sidebar sidebar"
            "footer footer";
    }
}
```

Conclusiones

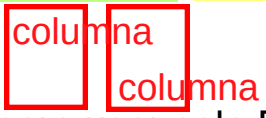
Como se ha podido ver durante todo el artículo, **no hay una solución mejor que otra**. El mejor sistema de trabajo por el que podemos optar es usar tanto Flexbox como CSS Grid en nuestros proyectos de forma conjunta, ya que **cada uno tiene sus puntos fuertes en diferentes cosas**.

- CSS Grid es mejor para la construcción general de nuestra página, ya que es más fácil modelar nuestra plantilla, y se pueden crear diseños más asimétricos y peculiares.
- CSS Grid en diseños con filas y columnas.



- Flexbox es mejor alineando el contenido dentro de los diferentes elementos de la web.

- Flexbox en diseños con filas o columnas. Funciona mejor en una dimensión.



Por todo esto, no hay motivos para usar solo Flexbox o CSS Grid. Te recomendamos aprender ambos y usarlos de manera conjunta

en tus proyectos.