

Fuente: <https://carlosazaustre.es/los-5-patrones-del-responsive-design/>

## Tiny Tweaks

Es el patrón más simple y sencillo de implementar de todos. Se basa en una sólo columna para el contenido.



Sus cambios son básicamente que dependiendo del tamaño de pantalla, se amplían los espaciados y el tamaño de fuente.

Es muy utilizado en sitios con mucho contenido escrito, así mejoran la experiencia de lectura.

Implementar este patrón en HTML y CSS sería así:

```
1<!DOCTYPE html>
2<html>
3<head>
4  <title>Tiny Tweaks</title>
5  <link rel="stylesheet" href="styles.css">
6</head>
7<body>
8  <nav>Navbar</nav>
9  <section class="columna1">Sección 1</section>
10</body>
11</html>

1/* Tiny Tweaks */
2.columna1 {
3  padding: 10px;
4  width: 100%;
5}
```

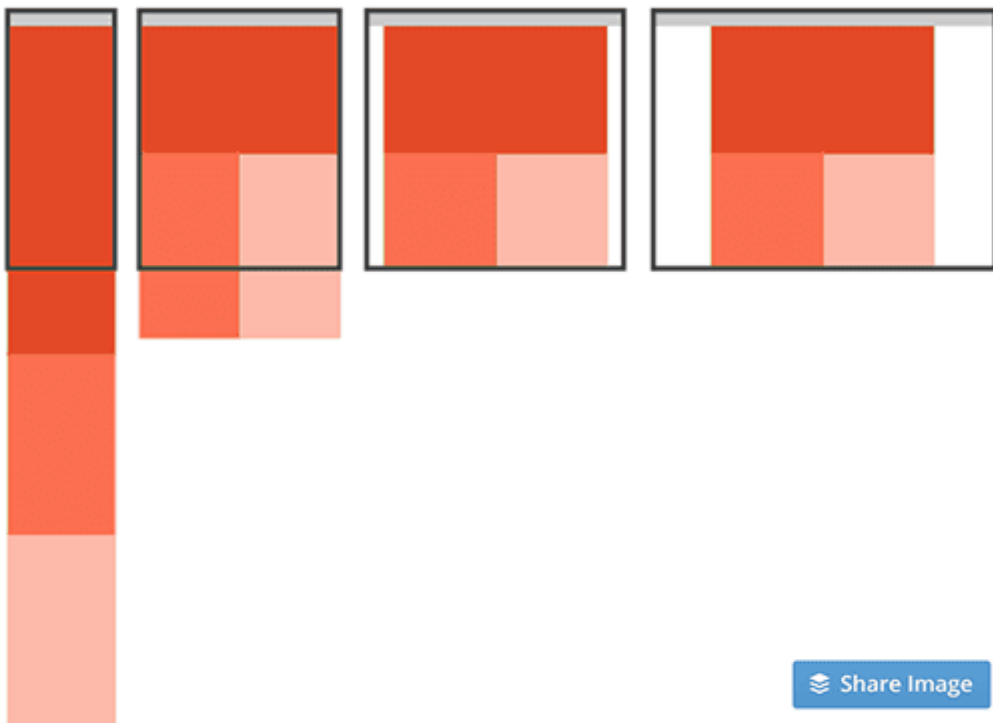
```


6
7 @media (min-width: 600px) {
8   .columna1 {
9     padding: 20px;
10    font-size: 1.5em;
11  }
12 }
13
14 @media (min-width: 800px) {
15   .columna1 {
16     padding: 40px;
17     font-size: 2em;
18   }
19 }

```

## Mostly Fluid

Este patrón consiste en tener una grilla o *Grid* de tamaño flexible.



 Share Image

Cuando estamos en un smartphone todo forma una única columna, y en varias filas quedan colocados los distintos bloques.

Según vaya creciendo la pantalla, los distintos bloques se agrupan ocupando toda la pantalla disponible.

En pantallas más grandes, el diseño es el mismo pero queda agrupado dentro de un contenedor que queda centrado en la página con un tamaño fijo de ancho.

Este patrón, implementado en HTML y CSS utilizando **Flexbox**, sería así:

```
1<!DOCTYPE html>
2<html>
3<head>
4  <title>Mostly Fluid</title>
5  <link rel="stylesheet" href="styles.css">
6</head>
7<body>
8  <nav>Navbar</nav>
9  <div class="container">
10   <section class="columna1">Sección 1</section>
11   <section class="columna2">Sección 2</section>
12   <section class="columna3">Sección 3</section>
13   <section class="columna4">Sección 4</section>
14   <section class="columna5">Sección 5</section>
15 </div>
16</body>
17</html>
1/* Mostly Fluid */
2.container {
3  display: -webkit-flex;
4  display: flex;
5  -webkit-flex-flow: row wrap;
6  flex-flow: row wrap;
7}
8
9.columna1,
10.columna2,
11.columna3,
12.columna4,
13.columna5 {
14  width: 100%;
15}
16
17@media (min-width: 600px) {
18  .columna2,
19  .columna3,
```

```

20 .columna4,
21 .columna5 {
22   width: 50%;
23 }
24
25
26 @media (min-width: 800px) {
27   .columna1 { width: 60%; }
28   .columna2 { width: 40%; }
29   .columna3,
30   .columna4,
31   .columna5 {
32     width: 33.3%;
33   }
34
35   .container {
36     width: 800px;
37     margin-left: auto;
38     margin-right: auto;
39   }
40 }

```

El bloque que contiene todos los bloques (.container) le añadimos la propiedad display: flex de *Flexbox* y que muestre el contenido en filas row.

Con wrap indicamos que si los bloques tienen que ocupar varias filas lo hagan.

Después con un par de *Media-Queries* disponemos dos breakpoints, uno para 600px simulando una tablet y otro para pantallas de más de 800px, simulando laptops y desktops de gran tamaño.

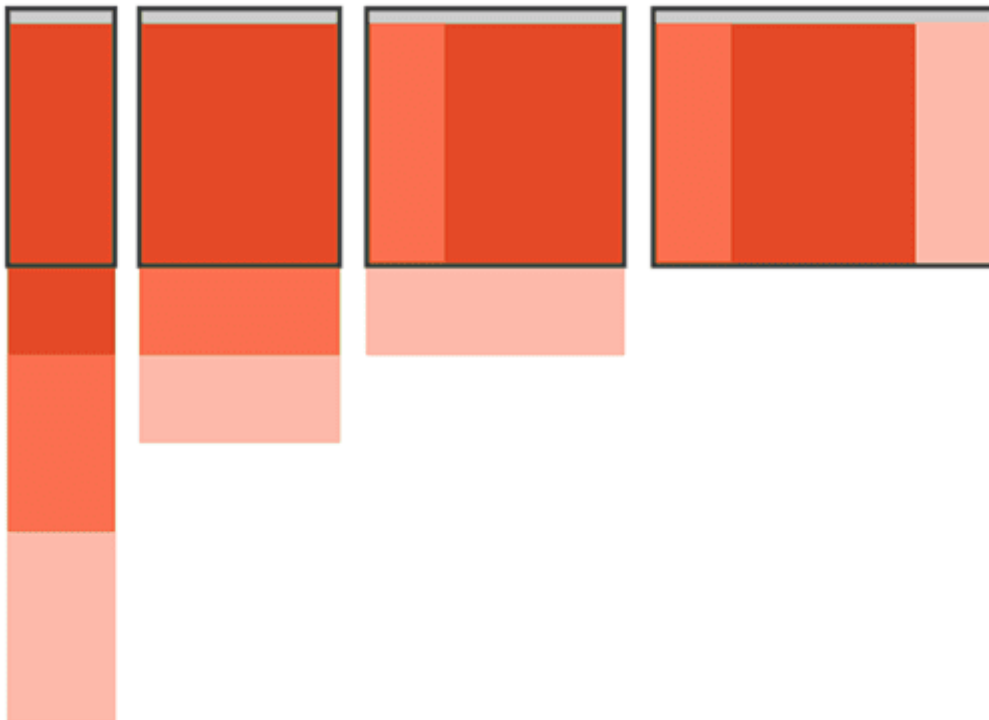
Para el tamaño *tablet*, hacemos que todas las columnas, salvo la primera, ocupen el 50%. De esta manera tenemos la primera columna ocupando todo el ancho de la pantalla y las restantes ocuparán 2 filas con dos bloques cada una.

En el *breakpoint* para pantallas de gran tamaño, hacemos que la primera columna y la segunda estén situadas en la primera fila, ocupando el 60% y el 40% del ancho respectivamente, y las tres restantes ocuparán 1/3 de la fila cada una.

En este momento, al `<div>` contenedor, le damos un ancho fijo de 800px para que el contenido quede centrado en la pantalla sin ocupar todo el ancho.

## Column Drop

Este patrón consiste en que cada bloque de contenido, que en un smartphone vemos en filas, vaya formando columnas según vaya siendo más grande la pantalla del dispositivo.



Tendremos un *primer breakpoint* donde la segunda fila pasa a ser columna, pero ocupando la primera posición, habitualmente para un menú de navegación.

El contenido que aparecía en primer lugar en la versión móvil, pasa a ocupar la segunda columna en el primer corte.

Tendremos un segundo punto de corte donde la última fila se convierte en columna también.

Este sería el código HTML y CSS para este patrón, utilizando Flexbox para ello:

```
1<!DOCTYPE html>
2<html>
3<head>
4  <title>Column Drop</title>
5  <link rel="stylesheet" href="styles.css">
6</head>
7<body>
8  <nav>Navbar</nav>
9  <div class="container">
10    <section class="columna1">Sección 1</section>
11    <section class="columna2">Sección 2</section>
12    <section class="columna3">Sección 3</section>
13  </div>
14</body>
15</html>
1/* Column Drop */
2.container {
3  display: -webkit-flex;
4  display: flex;
5  -webkit-flex-flow: row wrap;
6  flex-flow: row wrap;
7}
8
9.columna1,
10.columna2,
11.columna3 {
12  width: 100%;
13}
14
15@media (min-width: 600px) {
16  .columna1 {
17    width: 60%;
18    -webkit-order: 2;
19    order: 2;
20  }
21
22  .columna2 {
23    width: 40%;
24    -webkit-order: 1;
25    order: 1;
26  }
27
28  .columna3 {
29    width: 100%;
30    -webkit-order: 3;
31    order: 3;
```

```

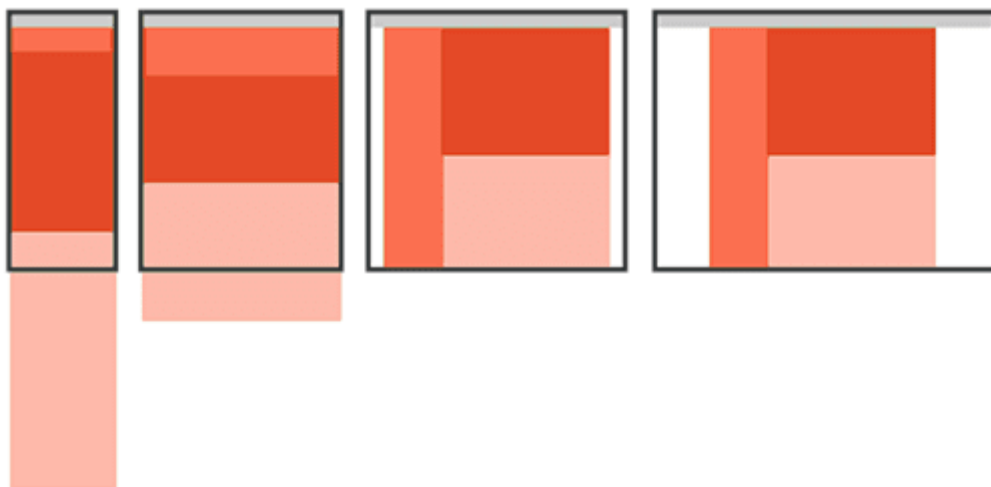
32 }
33}
34
35@media (min-width: 800px) {
36 .columna2,
37 .columna3 {
38   width: 20%;
39 }
40}

```

Para poder colocar con mayor facilidad la segunda fila en primera columna, hemos empleado la propiedad `order` de Flexbox, indicándole con un número que posición debe ocupar.

## Layout Shifter

Este es uno de los patrones más complejos. Consiste en mover los bloques de contenido cambiando totalmente el *Layout*, de ahí el nombre del patrón.



Gracias a Flexbox, estos cambios podemos realizarlos con mayor facilidad.

Las columnas 2 y 3 estarán englobadas dentro de un bloque que llamaremos *container-inner* que nos permitirá hacer el cambio de *layout*.

De nuevo, el código HTML y CSS para este patrón resultaría así:

```
1<!DOCTYPE html>
2<html>
3<head>
4  <title>Layout Shifter</title>
5  <link rel="stylesheet" href="styles.css">
6</head>
7<body>
8  <nav>Navbar</nav>
9  <div class="container">
10    <section class="columna1">Sección 1</section>
11    <div class="container-inner">
12      <section class="columna2">Sección 2</section>
13      <section class="columna3">Sección 3</section>
14    </div>
15  </div>
16</body>
17</html>
```

```
1/* Layout Shifter */
2.container {
3  display: -webkit-flex;
4  display: flex;
5  -webkit-flex-flow: row wrap;
6  flex-flow: row wrap;
7}
8
9.columna1,
10.columna2,
11.columna3,
12.container-inner {
13  width: 100%;
14}
15
16@media (min-width: 600px) {
17  .columna1 {
18    width: 25%;
19    height: 100vh;
20  }
21
22  .container-inner {
23    width: 75%;
24  }
25}
26
27@media (min-width: 800px) {
28  .container {
29    width: 800px;
```

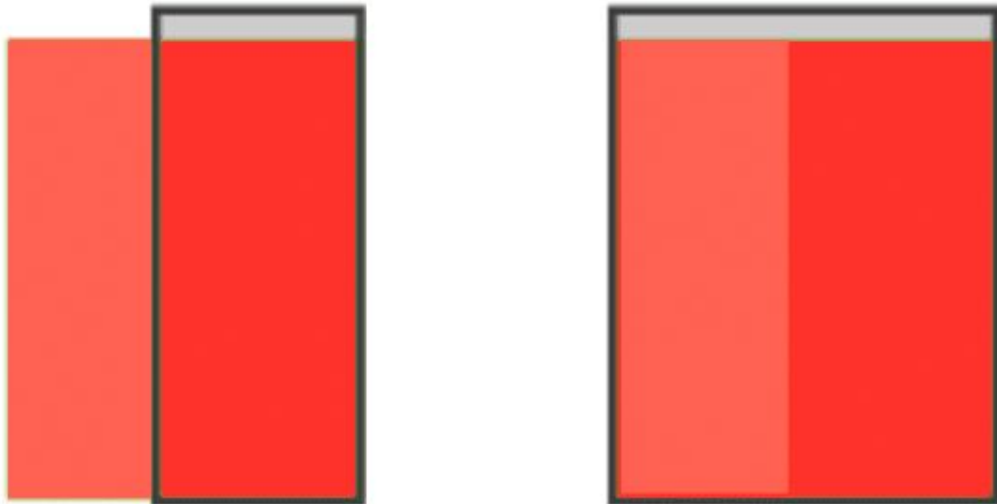


```
30 margin-left: auto;
31 margin-right: auto;
32 }
33 }
```

De nuevo, a partir del *breakpoint* de 800px, fijamos un ancho para el contenedor de manera que no ocupe toda la página.

## Off Canvas

Para el final he dejado el patrón más complejo de implementar pero uno de los más utilizados, sobre todo en aplicaciones móviles.



Este patrón esconde contenido en la pantalla y únicamente es visible si realizamos un determinado gesto. Este contenido oculto normalmente es un menú de navegación. Cuando la pantalla es más ancha, este contenido se hace visible.

El HTML para la página sería el habitual:

```
1<!DOCTYPE html>
2<html>
3<head>
4 <title>Off Canvas</title>
5 <link rel="stylesheet" href="styles.css">
6</head>
7<body>
```

```

8 <nav>Navbar</nav>
9 <div class="container">
10 <section class="columna1">Sección 1</section>
11 <section class="columna2">Sección 2</section>
12 </div>
13</body>
14</html>

```

Y el CSS vamos a verlo paso a paso

En primer lugar, la columna 1 que será la que aparezca oculta, le damos un ancho fijo, y le añadimos un position: absolute.

También le aplicamos una transición CSS a la transformación que vamos a declarar a continuación

```

1.columna1 {
2 position: absolute;
3 width: 250px;
4 height: 100vh;
5 -webkit-transition: -webkit-transform 0.3s ease-out;
6 transition: transform 0.3s ease-out;
7 z-index: 1;
8}

```

La transformación consiste en 2 fases. Para ocultar la capa, le aplicamos la transformación `translate(-250px, 0)` que lo que hace es "mover" la capa 250px hacia la izquierda en el eje X. Como el tamaño de la capa le hemos definido en 250px, quedará oculto

También creamos una clase `columna1 open` que añadiremos con JavaScript más adelante. Esta clase "mueve" a la posición 0 del eje X la capa, lo que hace sea visible.

Como hemos aplicado una transition este efecto será suave y otorga una mayor experiencia de usuario.

```

1.columna1 {
2 -webkit-transform: translate(-250px, 0);
3 transform: translate(-250px, 0);
4}
5
6.columna1.open {

```

```
7 -webkit-transform: translate(0, 0);
8 transform: translate(0,0);
9}
```

La columna 2 (central) no tiene mayor misterio. Queremos que ocupe el 100% del ancho y todo el alto de la página.

```
1.columna2 {
2 width: 100%;
3 height: 100vh;
4 position: absolute;
5}
```

Cuando la pantalla sea más ancha, vamos a aplicar al contenedor padre las propiedades de Flexbox. Esta vez el añadimos la propiedad nowrap que hace que el tamaño de ancho de los bloques de adapte en una sola fila, sin crear varias.

Ahora la columna que estaba oculta, la sacamos a la luz, aplicando la transformación de translate(0,0)

```
1@media (min-width: 600px) {
2 .container {
3 display: -webkit-flex;
4 display: flex;
5 -webkit-flex-flow: row nowrap;
6 flex-flow: row nowrap;
7 }
8
9 .columna1 {
10 -webkit-transform: translate(0, 0);
11 transform: translate(0, 0);
12 }
13}
```

Muy bien, ahora es momento de *aplicar magia* con JavaScript. Queremos que al hacer el típico gesto de *swipe* en el móvil hacia la derecha, aparezca el menú *off-canva*, y al hacer el gesto contrario se oculte hacia la izquierda.

Podemos usar los eventos nativos de JavaScript touchstart y touchmove pero el código puede quedar demasiado farragoso. Para solucionar esto tenemos la librería [Hammer.js](#) que tiene programados los diferentes gestos que podemos usar en un teléfono o tablet.

Para aplicar esto, lo primero que vamos a hacer es añadir unos id a los paneles en el HTML:

```
1<div class="container">
2  <section id="sidePanel" class="c1">Sección 1</section>
3  <section id="mainPanel" class="c2">Sección 2</section>
4</div>
```

sidePanel será el panel *off-canva* y mainPanel será el panel principal.

Añadimos también al HTML el script con la librería **Hammer.js**. Podemos usar una local descargada o un CDN como he puesto en el ejemplo.

```
1<script
src="https://cdnjs.cloudflare.com/ajax/libs/hammer.js/2.0.4/hammer.min.js"></script>
```

Y a continuación añadimos el código que detectará los eventos swiperight cuando arrastremos el dedo hacia la derecha y swipeleft cuando lo hagamos al contrario. Estos eventos irán adjuntos al elemento mainPanel que será el que los escuche.

Por tanto primero *cacheamos* los elementos sidePanel y mainPanel, y después se los añadimos a un objeto Hammer que lo gestionará.

Cuando el evento ocurra, simplemente queremos que se añada la clase de CSS open al sidePanel en el caso de que no esté añadida, y si no que la quite. Esto lo conseguimos con la función de JS toggle. El código JS final sería así:

```
1(function(){
2  var mainPanel = document.getElementById('mainPanel');
3  var sidePanel = document.getElementById('sidePanel');
4
5  var hammerPanel = new Hammer(mainPanel);
6
7  hammerPanel
8  .on('swiperight', function(e) {
```

```
9 sidePanel.classList.toggle('open');  
10 }  
11 .on('swipeleft', function(e) {  
12 sidePanel.classList.toggle('open');  
13 });  
14}());
```

---

Con esto tendríamos los 5 patrones de diseño Responsive implementados sin necesidad de frameworks. Todos ellos pueden combinarse entre sí, por ejemplo usar *Off-Canvas* con un *Column Drop* que use a su vez *Tiny Tweaks*. Todo es posible!