4.5 Conceptos y técnicas que aporta Responsive Web Design

4.5.1 El concepto del "Mobile First"

Hasta ahora todos los sitios web eran diseñados sólo para equipos de sobremesa y el proceso de navegar por la Web en los teléfonos móviles era bastante incómodo. Sin embargo, las tecnologías están cambiando y el principio de Mobile First se está convirtiendo en un concepto cada vez más extendido. Mobile First es una filosofía desarrollada por Luke Wroblewski, [20] la cual señala la priorización del entorno móvil frente al escritorio a la hora de desarrollar experiencias de usuario.

Mobile First se trata de la práctica de desarrollar un diseño desde su forma más básica planteándolo primeramente para la pequeña pantalla de un Smartphone, basándose en un crecimiento iterativo que aporte cimientos al diseño para otros dispositivos de mayor resolución.

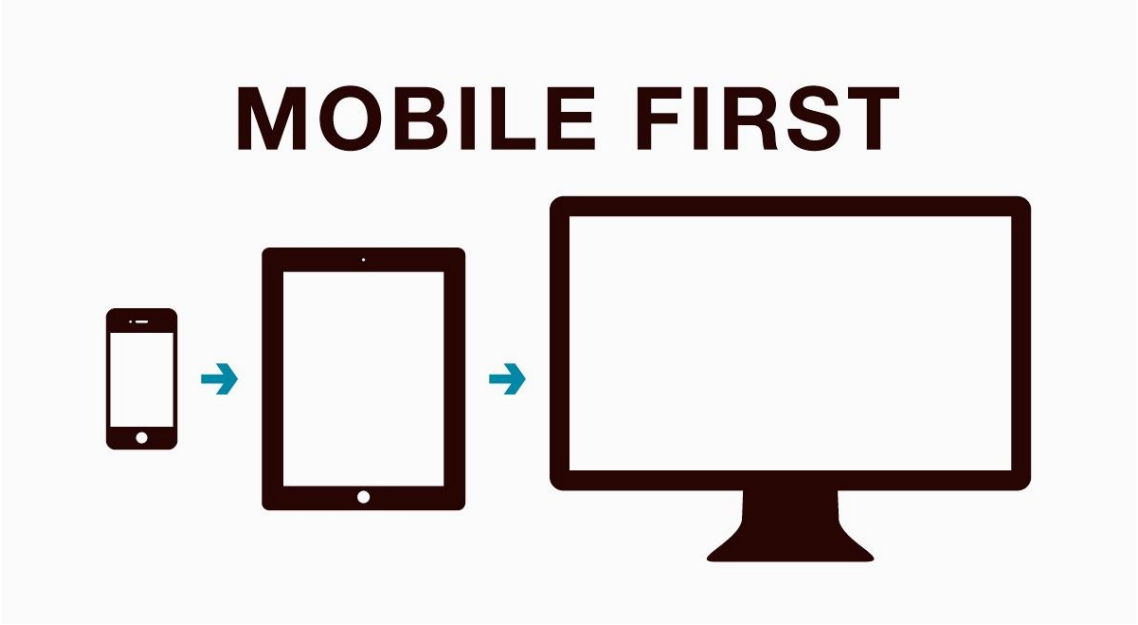


Ilustración 4 Mobile First

Esta técnica se centra en ir de lo más concreto a lo más abstracto centrándose únicamente en el contenido, implicando ir definiendo estilos gradualmente desde una simple base. La página web es creada inicialmente para dispositivos con menor capacidad y después se añaden nuevas características empleando Media Queries y CSS3. Los navegadores que no soporten esta tecnología recibirán el contenido simplificado (la versión mobile), y los navegadores avanzados trabajaran con las media queries definidos.

Los motivos esenciales para aplicar esta filosofía son los de llegar a más gente, obligar al diseñador centrarse en los contenidos y funcionalidades que son los que van a

marcar las necesidades del usuario aprovechando el espacio disponible y lo primordial con la tecnología que poseemos ahora mismo, sacar provecho de las funcionalidades móviles como pueden ser la geo localización o los eventos táctiles.

4.5.2 Layouts empleados en el diseño web

Un layout flexible es una de las mejores opciones para implementar un diseño web hoy en día, pero debemos de analizar otras opciones disponibles que son también destacables y que nos pueden ofrecer una decisión correcta dependiendo de las necesidades de nuestro proyecto.

En general se definen cuatro tipos de layouts, el layout de ancho fijo, el layout líquido o fluido, el elástico y el hibrido. Cada aproximación posee sus fortalezas, restricciones y propios retos.

4.5.2.1 Tipos de Layouts

Hasta ahora el layout de ancho fijo ha sido el más famoso y mayormente utilizado empleando la restricción de un ancho fijo especificado en pixeles, siendo 960 pixeles el tamaño de ancho comúnmente elegido. El numero 960 no es casualidad, es un numero amigable para la definición de layouts basados en grids, por ser divisible entre 3,4,5,6,8,10,12 y 15 ofreciendo gran variedad de opciones para ocupar el espacio de los grids.

La gran ventaja de estos layouts es que aumenta bastante la capacidad o sensación de tener bajo control el diseño, conociendo exactamente el ancho de cada elemento mostrando todos los componentes adecuadamente, siendo además aparentemente consistente a través de distintos tamaños de pantalla.

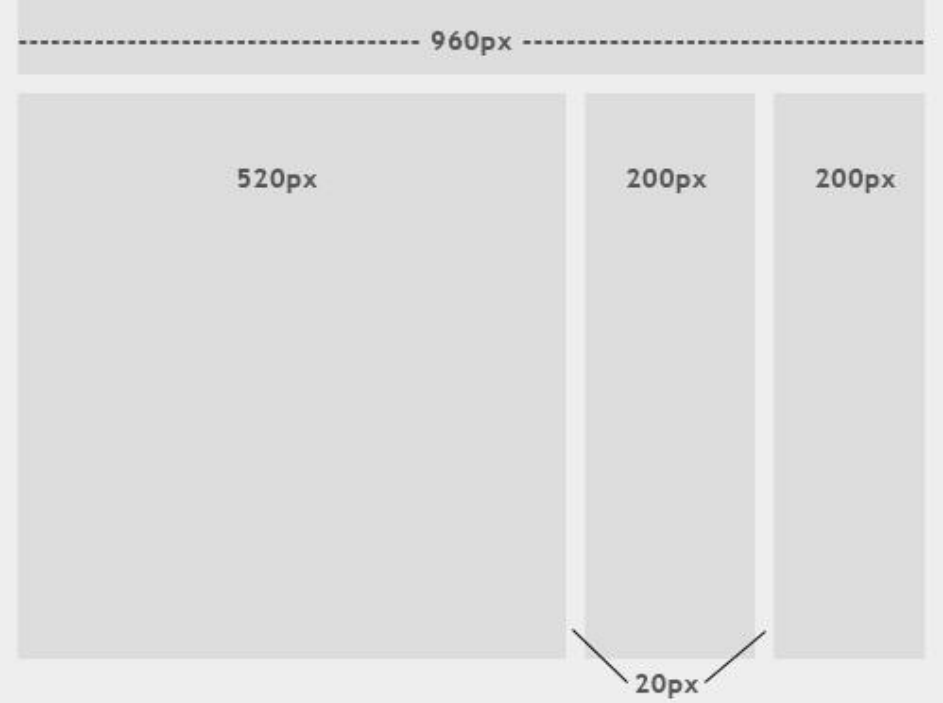


Ilustración 5 Layout de Ancho Fijo (Fuente: http://coding.smashingmagazine.com/)

El mayor problema de estos layouts es que se debe operar bajo ciertas suposiciones, cuando se determina cual debe ser la amplitud del sitio, se debe adivinar cuáles serán las dimensiones que mejor se adecuaran a la mayoría de nuestros visitantes. Esta decisión es bastante complicada y es la aparición de dispositivos como smartphones y tablets en nuestro tráfico hace de esta decisión un gran reto.

La consistencia que supuestamente derivan estos layouts es un poco engañosa ya que para pantallas más pequeñas que el amplio definido, el cliente solo verá una porción del sitio y por el otro lado si un usuario con una pantalla con mayor ancho se podrán ver a los lados una gran cantidad de espacio en blanco. Además la rigidez de estos layouts es un gran problema para la amplia diversidad de dispositivos. La mayoría de los navegadores móviles muestran un zoom alejado para que los diseños web quepan en pantalla ajustando su ancho, pero en este tipo de layouts se hace necesario acercarse para poder leer el contenido, lo que se aleja totalmente de una buena experiencia de usuario.

4.5.2.2 Layout Fluido

Los layouts fluidos son layouts cuyas dimensiones son determinadas por porcentajes y no por pixels. Como resultado se obtiene un diseño mucho más manejable. Por ejemplo podemos dividir un layout en tres columnas dividiendo el 100% de su espacio en un tamaño de 50% de ancho y 20% para las otras dos respectivamente. Usando este tipo de técnica definimos un layout para el cual no es necesario que el usuario emplee un navegador de ancho especifico, si no que el ancho de los elementos se ajustarán en forma de acordeón al ancho definido por el navegador del dispositivo ya sea un ordenador de escritorio, una tablet o un smartphone.

Este tipo de construcción de diseños evita muchos problemas que aparecían en los layouts de ancho fijo. Ya que el sitio se adapta al ancho disponible, el diseño puede ajustarse mejor a los espacios disponibles haciendo desaparecer esos espacios en blanco molestos que aparecían en los layouts de ancho fijo.

Implementando estrategias Responsive Web Design y apoyándose en media queries y CSS podemos optimizar la visión para diferentes resoluciones, haciendo que este tipo de layouts sea una buena estrategia. Sin embargo, también existen varios problemas que se deben solucionar ya que todo esto no es suficiente para asegurar que el diseño se vea adecuadamente tanto en un teléfono como en una televisión.

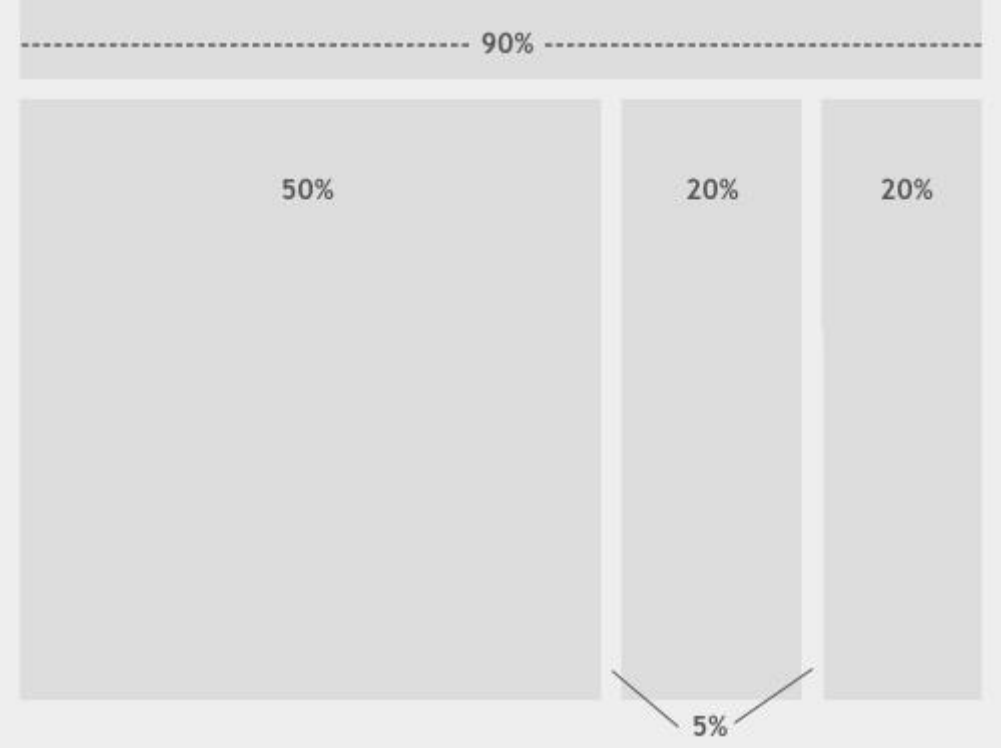


Ilustración 6 Layout Fluido (Fuente: http://coding.smashingmagazine.com/)

4.5.2.3 Layout Elástico

Los layouts elásticos se tratan de unos layouts similares a los layouts fluidos con la excepción de que estos tienen la restricción de ser controlados por el tamaño de la fuente, típicamente empleando la medida "em". Un "em" es el tamaño equivalente al tamaño predeterminado definido para la fuente. Por defecto el texto tiene un tamaño de fuente de 16px, por lo tanto un "em" correspondería con 16 pixeles calculando los tamaños proporcionalmente. Este tamaño por defecto puede cambiar dependiendo del navegador. Este tipo de layouts proveen un fuerte control sobre la tipografía. En general, los diseñadores recomiendan una longitud de línea para cada párrafo de entre 45 y 70 caracteres para que proporcione una buena experiencia buena al usuario a la hora de leer el texto.

Más beneficios de este tipo de layouts es que los usuarios pueden incrementar y decrementar el tamaño de la fuente escalando los elementos en proporción al tamaño de ésta, aunque por el contra, se puede generar exceso de contenido vacío que disminuye el aspecto estético del diseño web.

4.5.2.4 Layouts Híbridos

La opción de los layouts híbridos combina las ideas de los layouts descritos anteriormente. Un ejemplo puede ser mantener una columna con un ancho fijo, pero emplear medidas relativas para el resto de columnas. Estas técnicas por ejemplo se emplean cuando se diseñan layouts que necesitan elementos como anuncios con un tamaño especificado en una de sus columnas y ocupar el resto del espacio con porcentajes. Sin embargo, no es una opción muy empleada, por la dificultad de implementación mezclando cálculos relativos con absolutos.

4.5.3 El problema del modelo de caja

El modelo de caja por defecto de CSS trabaja añadiendo el padding y el borde definido al elemento correspondiente. Esto quiere decir que aunque nosotros definamos un ancho y un alto determinado, esto no se respetará si el elemento contiene tanto un padding o un borde que serán añadidos al modelo de caja del elemento, por lo tanto se renderiza en el navegador todo elemento visible dentro del modelo de caja.

Para muchos este tratamiento del modelo de caja no es nada intuitivo. Esto puede ser un inconveniente al trabajar con porcentajes por lo que existe una solución que nos aporta CSS3 para trabajar con los modelos de caja.

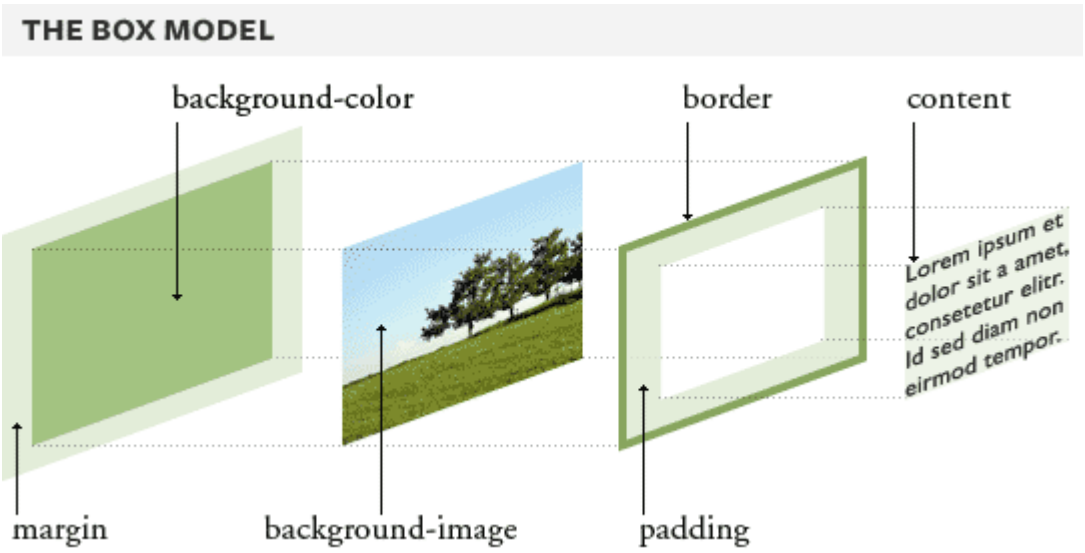


Ilustración 7 The Box Model (Fuente: http://hisakoevans.com/web-tutorial/div.html)

Esta solución que incluye CSS3 es la propiedad box-sizing, la cual es capaz de cambiar el modelo de caja a modelos más intuitivos que nos ayudarán a definir nuestros anchos en diseños fluidos.

Una opción definida para el atributo box-sizing es border-box. Este valor transforma el modelo de caja haciendo que el ancho especificado sea equivalente al ancho total de la caja que hemos determinado en la hoja de estilos. Es decir, si hemos especificado un ancho de 200 px para el elemento, el padding y el border se mantendrán pero los elementos interiores se redimensionaran proporcionalmente para que, finamente, el elemento el cual hemos determinado su ancho siga manteniéndose. Esto es interesante cuando determinamos columnas con un porcentaje determinado ya que ocupar 1 pixel más destrozaría toda la estructura.

Veamos un ejemplo de esta propiedad en acción. Supongamos que tenemos dos capas cada una con un identificador para aplicarles el estilo predeterminado:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8"/>
		<meta name="viewport" content="widht=device-width"/>
		<title> Box Sizing</title>
		<link rel="stylesheet" href="style.css"/>
	</head>
	<body>

		<div id="con-border-box">
			Con Border Box
		</div>
		<div id="sin-border-box">
			Sin Border Box
		</div>
	</body>
</html>
```

A la primera de las cajas vamos a determinarle un ancho y un alto de 200px, además queremos determinar un padding y un borde, ambos de 10 px. Apliquémosle el atributo box-sizing para cambiar nuestro modelo de caja a border-box:

```
#con-border-box{
	background-color: red;
	-webkit-box-sizing: border-box;
	-moz-box-sizing: border-box;
	box-sizing: border-box;
	width: 200px;
	height: 200px;
	padding: 10px;
	border: 10px solid #ccc;
}
```

Si tomamos la segunda caja y aplicamos las mismas reglas pero sin cambiar el modelo de caja. Veamos que sucede en el navegador:

```
#sin-border-box{
	background-color: red;
	width: 200px;
	height: 200px;
	padding: 10px;
	border: 10px solid #ccc;
	margin-bottom: 10px;
}
```

El resultado en el navegador lo podemos apreciar en la siguiente ilustración:

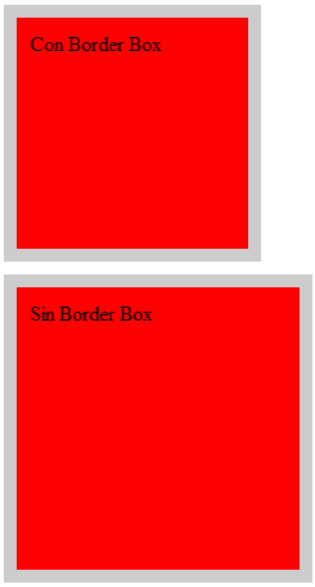


Ilustración 8 Resultado con cada modelo de caja

Como podemos observar cada elemento ha tomado dimensiones diferentes. Para el primer elemento, como vemos en la ilustración, se han conservado el padding y el borde que les hemos determinado. Sin embargo, el interior se ha redimensionado para que al final el total que ocupa todo el modelo sea 200px, es decir 160px mas 10px a cada lado de relleno más otros 10 px a cada lado de borde.

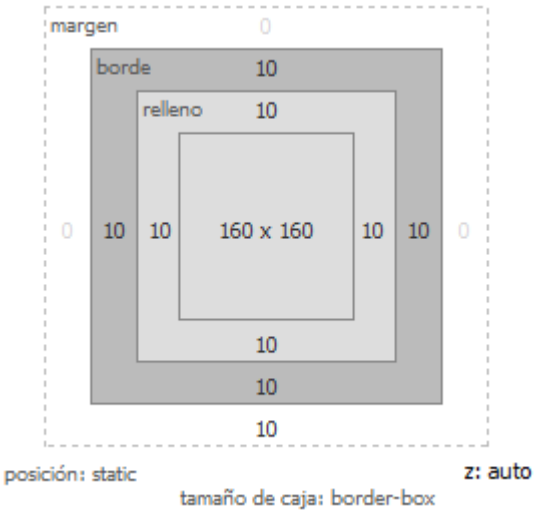


Ilustración 9 Modelo de Caja con Border-Box

En cambio, para el segundo elemento, los 200 px establecidos se han mantenido añadiendo externamente el padding y el borde que hemos determinado en ambas situaciones.

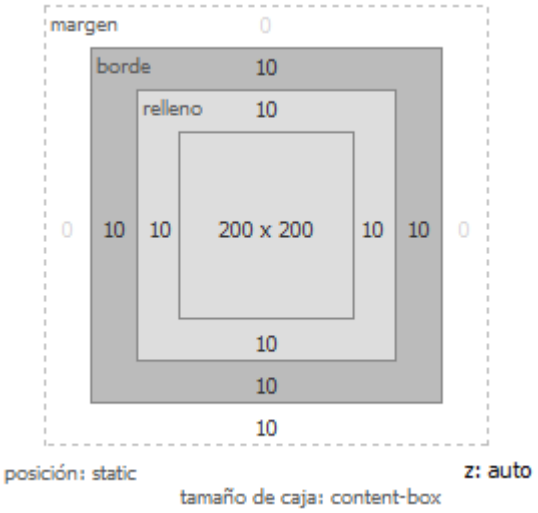


Ilustración 10 Modelo de Caja con Content-Box

4.5.4 Los Mediaqueries y CSS3

La nueva versión de CSS ha evolucionado el concepto de los mediaqueries. Hasta ahora los mediaqueries habían sido empleados para mostrar la presentación en dispositivos como impresoras. Sin embargo, con esta nueva versión se emplea este concepto un paso más allá para detectar el tipo de dispositivo conectado a la web, consiguiendo la posibilidad de adaptarse al dispositivo concreto a través de distintos factores como pueden ser ancho y alto de la ventana del navegador, ancho y alto del dispositivo, la resolución del dispositivo o la orientación de la pantalla del dispositivo.

A la hora de utilizar los mediaqueries debemos de tener en cuenta el ancho de ventana del navegador y no la resolución de pantalla del dispositivo. Por ejemplo si definimos una mediaquerie como la siguiente:

```
<link rel="stylesheet" media="screen and (min-device-width: 960px)" href="960.css" />
```

El atributo min-device-width tiene en cuenta la resolución del dispositivo a la hora de ejecutar el CSS, esto quiere decir que si reducimos el ancho del navegador, seguirá mostrándose de la misma manera ya que la resolución seguirá manteniéndose, por lo que no se adaptará al nuevo ancho de ventana. Por lo tanto, lo más adecuado es parametrizar estas acciones teniendo en cuenta el ancho de ventana del navegador y no de la resolución del dispositivo.

Por ejemplo tomemos la siguiente mediaquerie:

```
@media screen and (min-width: 960px) {
    /* Código CSS*/
}
```

Esta mediaquerie es una expresión condicional por el cual si se cumplen las condiciones que se especifiquen se aplicará el código CSS que se encuentre en su interior. La primera condición, en este ejemplo screen, indica el tipo de dispositivo que mostrará la salida, por ejemplo, una impresora o una pantalla y el segundo parámetro nos indica el ancho de ventana, para este ejemplo el ancho de pantalla ha de ser como mínimo 960 pixeles para que se interprete el código.

Además se pueden incorporar varias condiciones, por ejemplo, podemos especificar un máximo y un mínimo para el ancho de pantalla:

```
@media screen and (min-width: 480px) and (max-width: 960px) { }
```

Para esta mediaqueries se ejecutará el código CSS que tengo un ancho de pantalla entre 480 pixeles y 960 pixeles ambos incluidos.

Además de las características para determinar las resoluciones y anchos de pantalla, podemos determinar otros parámetros, como por ejemplo la orientación del dispositivo, importante en dispositivos móviles:

```
@media screen and (orientation: landscape) { }
```

Una vez que conocemos como crear puntos de ruptura en nuestros diseños, la pregunta que debemos de realizarnos es como determinamos en que puntos concretos debemos de crear un nuevo punto de ruptura.

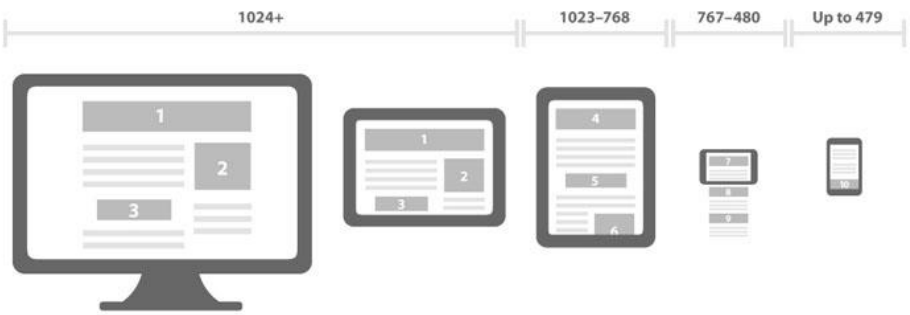


Ilustración 11 Mediaqueries (http://blog.bitmarketing.es/por-que-usar-responsive-web-design/)

La forma convencional de determinar estos saltos ha sido teniendo en cuenta anchos determinados de ciertos dispositivos, por ejemplo 320px que es el ancho del iPhone, 768px que es el ancho del iPad y 1024px soportado por la mayoría de los equipos de escritorio. Confiar en unas medidas estándar para definir unos puntos de ruptura por defecto no es realmente adecuado.

Si empezamos a definir puntos de ruptura teniendo en cuenta resoluciones de dispositivo para llegar a un concepto común, corremos un riesgo bastante alto, ya que el mercado de dispositivos está en constante crecimiento, apareciendo nuevos dispositivos con diversas características en lo que a tamaños de pantalla se refiere. Por lo tanto, es una batalla perdida, la mejor aproximación es usar la propia intuición ya que el mismo diseño pedirá cuando establecer el siguiente salto y reorganización de los elementos.

4.5.5 Concepto de Viewport

El viewport se define como la zona visible de un navegador, es decir se limita por el ancho del navegador. Sin embargo, todo cambia al mostrarse en un dispositivo móvil. A pesar de este tipo de pantalla, los navegadores por defecto intentan mostrar el sitio completo con el objetivo de proveer una experiencia web completa, pero esto no es totalmente adecuado.

La mayoría de los navegadores móviles escalan las páginas HTML al ancho del viewport para que se pueda ver completa en la pantalla. Sin embargo, HTML5 nos propone una meta tag para controlar esta acción. El tag viewport nos permite decir al navegador que emplee el ancho del dispositivo como ancho de viewport y así además se puede desactivar la escala inicial por defecto del navegador. Esta meta tag se debe emplear en las cabeceras html dentro de la etiqueta <head>.

Básicamente gracias a esta función vamos a indicar como debe de comportarse el navegador a la hora de mostrar en su pantalla nuestro diseño web. Un ejemplo de uso de esta etiqueta es el siguiente:

```
<meta name="viewport" content="width=device-width, initial-
                scale=1, maximum-scale=1">
```

Por defecto los navegadores muestran un ancho determinado de página dependiendo del terminal y el navegador, por ejemplo, en el caso de iPhone el ancho por defecto es de 980 pixeles. Esto tiene un inconveniente, porque además de mostrarse la página web reducida no se aplicarán los mediaqueries definidos. Gracias a esta etiqueta podemos establecer parámetros como el ancho, el alto o el zoom de la pantalla.

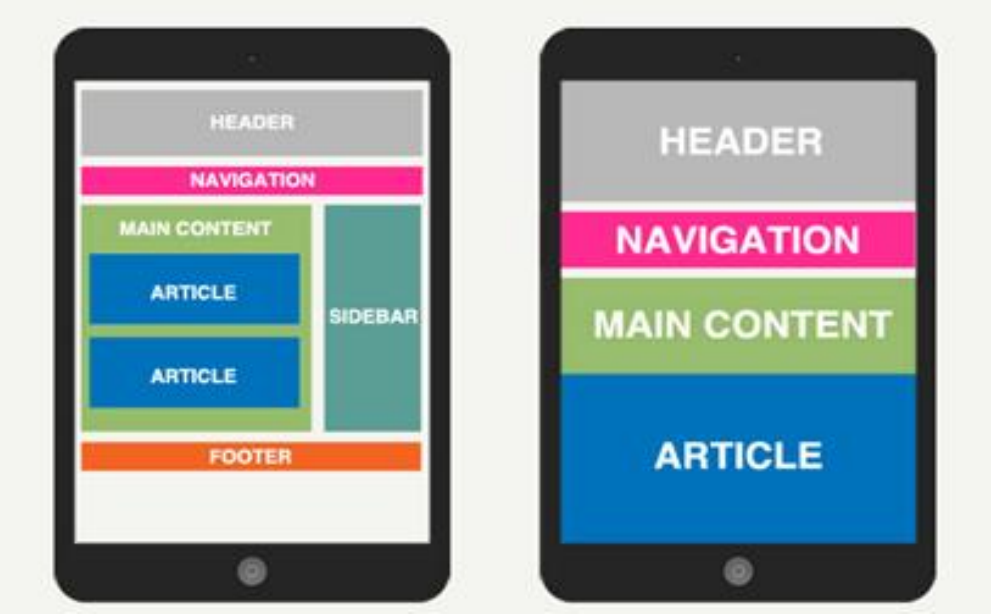


Ilustración 12 Estableciendo el viewport

La meta viewport posee una lista de parámetros clave-valor separadas por comas. Un ejemplo de las distintas propiedades aplicables son las siguientes:

```
<meta name="viewport" content="
width = [pixels | device-width ],
height = [pixels | device-height],
initial-scale = float,
minimum-scale = float,
maximum-scale = float,
user-scalable = [yes | no] ">
```

A continuación se detalla cada una de estas propiedades:

Width

Con la propiedad width podremos indicar el ancho que tomará la página renderizada en el dispositivo. Esta propiedad nos permite indicar un valor estático en pixeles para páginas con contenido fijo, pero para el objetivo de este proyecto podremos indicar el valor device-width, que representa el ancho real del dispositivo, adaptándose la web al viewport y así pudiendo llevar a cabo las media queries.

Height

La propiedad height representa al igual que la propiedad anterior una medida de la pantalla, en concreto indica el alto, aunque esta propiedad no posee demasiada utilidad funciona exactamente igual que la propiedad width, poseyendo el parámetro device-height por el cual se toma el alto de la pantalla del dispositivo.

Initial-Scale

La propiedad initial-scale nos indica el zoom inicial que se está aplicando a la página. Si has visitado alguna página desde un dispositivo móvil sin Responsive Web Design podrás ver como es necesario realizar zoom para poder leer el contenido o acercarte a cierta parte de la web. Gracias a esta propiedad podemos regular la escala de zoom. Este parámetro nos permite indicar el tanto por cierto, por lo que si ponemos un valor de 0,9 se podrá ver inicialmente un zoom del 90% respecto el ancho total.

Maximum/Minimum Scale

Al igual que con la propiedad anterior podríamos indicar el zoom por defecto inicial, con estas dos propiedades podemos indicar cuál será la escala máxima y mínima a la hora de establecer zoom.

User-Scalable

Esta propiedad está directamente relacionada con la anterior, ya que podemos controlar la acción del usuario a la hora de hacer zoom. Con un valor booleano

lograremos indicar si deseamos que el usuario manipule el zoom para visualizar nuestra web.

Respecto al tema de la etiqueta viewport la comunidad web considera de malas práctica el empleo de las etiquetar user-scalable y minium-scale, ya que por ejemplo se considera un gran problema de accesibilidad a personas con problemas de visión el impedir no poder moverse en la página con total libertad o limitando el zoom.

4.5.6 El empleo de medidas relativas

Hasta ahora los diseños webs tomaban medidas absolutas en pixeles basadas en resoluciones concretas. Por otro lado, con esta filosofía se emplean porcentajes y unidades como los "em" en los diseños con el objetivo de mantener los elementos tan flexibles como sea posible. Las unidades relativas nos acercan a este concepto, proporcionándonos herramientas para adaptarnos a las enormes variaciones de los tamaños de pantalla, densidad de pixeles y los niveles de zoom.

Mientras las Media Queries son el secreto esencial de las webs responsive, para la mayor parte del trabajo vamos a tener que trabajar con estas unidades de medida. Usando un grid flexible, debemos olvidarnos de anchos fijos en los elementos. Esto significa que todos los elementos de nuestro diseño modifican su tamaño proporcionalmente entre sí. La idea básica de un grid fluido es utilizar la fórmula que nos provee Ethan Marcotte para convertir dimensiones absolutas basadas en pixeles en unidades relativas.

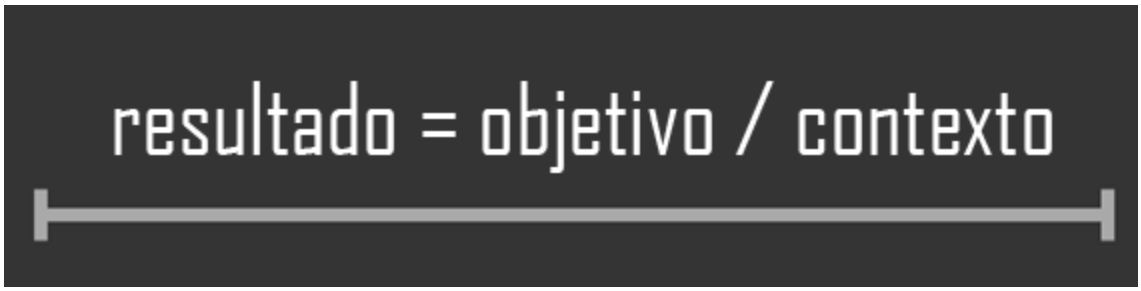


Ilustración 13 Fórmula de Ethan Marcote

Por ejemplo si poseemos un layout de 800pixeles de ancho con dos columnas, una de ellas 580px y la otra de 220px, y deseamos transformar estas medidas a porcentajes debemos de dividir el objetivo entre el contexto padre, para este ejemplo:

```
580/800=72,5%
220/800=27,5%
```

En el código CSS se deben emplear estos porcentajes:

```
#column1 {    width: 72.5%;  }
#column2 {    width: 27.5%;  }
```

Tan solo debemos cambiar la manera de pensar y empezar a tomar las medidas necesarias en porcentajes, empleando unos sencillos cálculos siempre teniendo cuidado

de no escoger el contexto equivocado por lo que debemos tomar la referencia adecuada, el elemento del cual toma la referencia.

4.5.7 Imágenes flexibles

Uno de los mayores problemas que son necesarios resolver con un diseño Responsive es trabajar con imágenes. Hay diversas técnicas para redimensionar imágenes proporcionalmente y muchas de ellas son fáciles de implementar. Sin embargo, la opción más popular descrita por Ethan Marcote es el uso del atributo max-width.

```
img { max-width: 100%;}
```

Con esta técnica indicamos al navegador que el ancho de la imagen nunca debe superar el ancho de su contenedor padre, por lo tanto, gracias a este atributo nunca se desbordarán del elemento padre sin romper nuestro diseño, Por lo cual, si el elemento padre determina su ancho respecto al ancho del navegador, la imagen se redimensionará para dar una sensación única de flexibilidad.

Además de ser útil para imágenes también tiene bastante utilidad en otros tipos de objetos multimedia. Por ejemplo, si queremos embeber un video de cualquier fuente como Youtube pero queremos que este se adapte al tamaño, también podemos aplicarle la misma regla:

```
Img, object, embed, video { max-width: 100%}
```

Veamos a continuación un ejemplo donde encapsulamos una imagen dentro de una capa div denominada wrapper.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8"/>
		<meta name="viewport" content="widht=device-width"/>
		<title> Imágen Fluida</title>
		<link rel="stylesheet" href="style.css"/>
	</head>
	<body>

		<div id="wrapper">
			<img src="bewater.jpg">
		</div>
	</body>
</html>
```

Este wrapper vamos a asignarlo al centro de la pantalla y vamos a asignarle un ancho relativo de 80% respecto del ancho del navegador:

```
#wrapper
{
	width: 80%;
	margin: 0 0;
}
```

Dentro de esta capa se encontrará la imagen la cual vamos a asignar un ancho máximo del 100% disponible:

```
img
{
	max-width: 100%;
}
```

De esta manera podremos comprobar como la imagen se adapta al tamaño del contenedor:

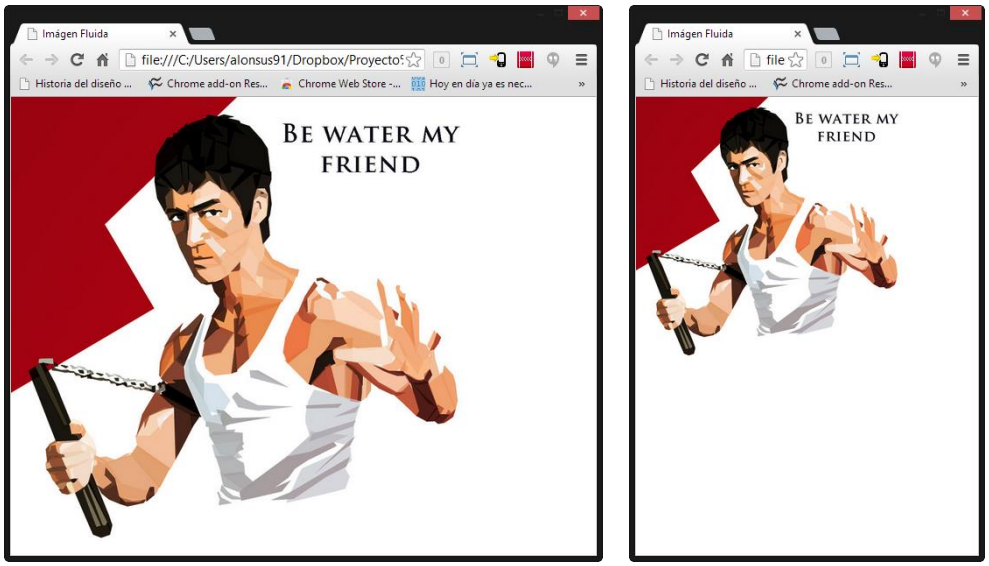


Ilustración 14 Ejemplo Imagen Fluida

Sin embargo, la técnica de las imágenes fluidas tiene sus desventajas, ya que aunque la imagen se redimensione en dispositivos de menor tamaño, el peso de las imágenes es el mismo que el de la imagen al 100%, por lo tanto la carga de la imagen en un dispositivo móvil puede llegar a ser un problema.

La solución a este problema sería cargar una imagen con distintas resoluciones y que se cargue dinámicamente dependiendo del tamaño de pantalla. Sin embargo, no es nada sencillo, existen distintas implementaciones y soluciones por partes de la comunidad, pero ninguna es la panacea.

Los desarrolladores web utilizan una variedad de hacks para evitar el problema empleando cachés y otras técnicas para mejorar en rendimiento, pero para que realmente se resuelva el problema, la web necesita nuevas herramientas por lo que el W3C está trabajando en una posible solución muy próxima. [21]

W3C ha publicado un borrador de la posible solución:

```
<picture alt="Description of image subject.">
        <source srcset="small.jpg 1x, small-highres.jpg 2x">
        <source media="(min-width: 18em)" srcset="med.jpg 1x, med-
highres.jpg 2x">
        <source media="(min-width: 45em)" srcset="large.jpg 1x, large-
highres.jpg 2x">
        <img src="small.jpg" alt="Description of image subject.">
</picture>
```

Básicamente la idea de este código sería determinar que se va a cargar una imagen y darle distintas fuentes para consumir. Además de determinarle en que breakpoint debe cargarse la imagen, se podría elegir la imagen dependiendo de la definición que permite el móvil, si se trata de una resolución estándar o para pantallas dpi. La última línea emplea la etiqueta img estándar para ser compatible con navegadores más antiguos.

Pero esta solución de momento es una propuesta del W3C que se encuentra en desarrollo, cuando HTML5 se transforme en un estándar es posible que esta solución esté disponible y evitemos todo este tipo de problemas.

4.5.8 Fuentes flexibles

Las fuentes flexibles están estrechamente relacionadas con los layouts elásticos. Para que nuestro diseño sea totalmente fluido al dispositivo, además de los anchos de los elementos, deben de graduarse los tamaños de fuente de los textos. Como se ha comentado los ems son la medida adecuada, ya que esta unidad, es al tamaño de fuente lo que los porcentajes son a los pixeles.

Por defecto cada navegador tiene definido un tamaño de fuente predeterminado, si determinamos el atributo font-size al 100% estamos diciendo que la fuente tendrá un tamaño respectivo al 100% con su tamaño de fuente relativo. Por lo tanto, tomará su medida relativa. Si nuestro texto se encuentra dentro de la etiqueta body, el tamaño relativo que toma es el de navegador y en algunos casos puede ser un tamaño de 14 a otros que puede ser 16 pixeles. Lo que concuerda en todos los navegadores es que el 100% del tamaño corresponde a 1em en ese navegador. Si nosotros determinamos el tamaño predeterminado de la fuente a otra medida mayor, por ejemplo 24 pixeles, 1 em correspondería a esa medida.

Se recomienda a la hora de desarrollar diseños Responsive acostumbrarse a definir fuentes en ems, y calcular los tamaños determinados a través de la fórmula de Ethan Marcote. Una buena práctica es definir en la etiqueta body el tamaño base de la fuente, por ejemplo determinar el tamaño por defecto del navegador:

```
body{ font-size: 100%}
```

Con esta decisión esta será la referencia para el resto de elementos del diseño.

Acompañémoslo de un ejemplo. Supongamos que tenemos un párrafo en el body el cual queremos que el tamaño de fuente sea 14 pixeles. Como por defecto el body tiene un tamaño del 100% que corresponde a 16 px debemos de aplicar la fórmula de Ethan Marcote al párrafo. En este caso el contexto son 16 pixeles y el objetivo 14 pixeles.

```
14px/16px= 0.875 em
```

Por lo tanto en nuestro CSS debemos de definir lo siguiente para que nuestra fuente sea proporcional:

```
p { font-size: 0.875em}
```

Ahora supongamos que dentro de este párrafo queremos destacar ciertas palabras usando la etiqueta empleando un tamaño de 18 pixeles. En este caso el contexto de la etiqueta span dentro de un párrafo es los 14 pixeles que habíamos definido para el párrafo.

```
<body>

      <p> Responsive <span>Design</span> mola! </p>
</body>
```

Por lo tanto

18px/14px = 1.2757em

```
p span{ font-size: 1.2857em }
```

Veamos cómo queda en el ejemplo final.

Responsive Design mola!

4.5.9 Técnicas para mostrar y ocultar contenido

Una de las técnicas empleadas en Responsive Web Design es la de ocultar o mostrar elementos según el tipo de resolución. Aunque hemos visto técnicas para transformar proporcionalmente los elementos y reorganizarlos, muchos desarrolladores emplean esta técnica para proporcionar una navegación más simple, un contenido más específico o simplemente privar de la funcionalidad a determinado dispositivo. Favorablemente CSS nos permite mostrar u ocultar elementos, pero vamos a determinar que técnica es la más apropiada.

En CSS tenemos la opción display, la cual podemos establecer en none, así el elemento HTML al que se le aplique esta propiedad se verá oculto. Podemos establecerlo

tanto de forma predeterminada o asociársela a un elemento dinámicamente a través de JavaScript.

```
display:none
```

Esta técnica se emplea tanto para ocultar o mostrar contenido en resoluciones pequeñas como grandes, un ejemplo de ello es cambiar el tipo de navegación ocultando un bloque y haciendo mostrar otro.

Por otro lado, existe otra opción en CSS llamada visibility, que se puede establecer a valor hidden. Sin embargo, esta opción esconde el contenido pero el elemento permanece en la misma posición a diferencia de display:none que oculta totalmente como si no estuviera en el DOM.

Con esta habilidad para manejar la visibilidad de los elementos nos abre las puertas a distintas transformaciones, adaptar nuestros diseños, reorganizar elementos y abrir la mente del diseñador para mostrar al usuario lo que realmente se desea mostrar

4.5.10 Rendimiento Web

El rendimiento Web es uno de los factores que más preocupan a los desarrolladores, sobre todo al trabajar con la filosofía Responsive Web Design, ya que transmitir la página web a un dispositivo móvil requiere un coste que las conexiones 3G deben asumir a pesar de no proporcionar la misma velocidad que una red cableada. Por lo tanto, no todo es la forma en que se ve, si no también cuenta el tiempo, para muchos oro y para los clientes la decisión de permanecer en la web o abandonarla.

Existen multitud de herramientas online y plugins para analizar los tiempos de carga y optimizar nuestros recursos para lograr la experiencia de usuario más cómoda. Entre ellos se encuentran GTMetrix (http://gtmetrix.com/), Firebug (http://getfirebug.com/) o Pingdom (http://tools.pingdom.com/fpt/). Estas herramientas evalúan el rendimiento de nuestra web y nos proporcionan un informe de los puntos que se debemos mejorar aportando ciertas recomendaciones.

Entre las razones de optimizar los tiempos de carga nuestras páginas webs se encuentran mejorar la experiencia de usuario causando una impresión positiva en el cliente o minimizar los problemas de latencia por las distancias entre los clientes los servidores.

Desde el punto de vista empresarial usar buenas prácticas de optimización hace al desarrollador más profesional y excelente desde la visión del cliente, lo que permite incrementar los ingresos y beneficios.

A continuación detallare buenas prácticas y consejos a la hora de desarrollar nuestra página web para ganar en tiempos de carga [22]:

Aunque se ha recomendado la técnica de la visibilidad de los elementos con el parámetro display, no es una buena opción rellenar nuestro código de display:none, ya que aunque el elemento contenga este atributo y no se visualice igualmente se descargará todo el contenido y se incluirá en el código por lo que los tiempos de cargas serán mayores.

También se recomienda incluir las hojas de estilos CSS y los documentos JS en un único fichero, ya que cada fichero necesitará una nueva petición HTTP por parte del navegador incrementando los tiempos de carga. Asimismo se recomienda que estos ficheros sean externos, y evitar incluir etiquetas CSS o styles o embeber código javascript en el HTM. La razón principal es que estos ficheros se guardan en la caché del navegador, y no es necesario volver a descargarlos, aunque esto puede ser un inconveniente mientras se está desarrollando, borrando repetidas veces la caché para probar el nuevo código.

Además del tema de la caché encapsulando código en ficheros externos, evitamos la repetición de código aligerando el documento html. Respecto al tema de aligerar el peso también es recomendable minimizar u ofuscar el código css y javascript, por ejemplo una herramienta que ayuda a esta tarea es Minify (http://code.google.com/p/minify/).

Se recomienda poner los enlaces a las hojas de estilos en el header ya que los navegadores por defecto no cargan la página web hasta haber obtenido las hojas de estilos manteniendo el viewport en blanco. Por otro lado, los JavaScript se recomiendan ponerlos después del body, justamente por lo contrario, para poder visualizar la página cuanto antes correctamente mientras se termina de descargar los scripts correspondientes.

Además se debe de que tener en cuenta que los redireccionamientos son un gran inconveniente para la optimización, ya que estos aumentan potencialmente los tiempos de espera.

4.5.11 Compatibilidades entre navegadores

4.5.11.1 Concepto de Cross Browser

El término Cross Browser se refiere al diseño de páginas webs que se ven y se comportan exactamente igual sin importar el navegador que las esté interpretando. Este punto se debe de tener muy en cuenta a la hora de diseñar Responsive Web Design, ya que lo que aparentemente funciona o se ve de tal modo en un navegador web, puede romper nuestro diseño en cualquier otro. Se trata de una barrera más que debemos de afrontar e intentar que pase por desapercibido.

Actualmente existen multitud de navegadores desarrollados por distintas empresas de software, los cuales difieren en bastantes puntos. Estos navegadores deberían de basarse en las reglas que organismos como el W3C determinan para una única interpretación del lenguaje HTML, CSS o JavaScript. Sin embargo, en algunos casos no son respetadas o lo peor que ciertas etiquetas no son soportadas en algún navegador. Este problema es el que reside todavía HTML5 todavía no convertido como estándar, aunque se prevé para 2014. [23]

Por lo tanto, debemos de abordar este problema proporcionando soluciones que se adapten a todo tipo de navegadores. Los problemas que pueden surgir provienen de la interpretación del código y etiquetas. Por ejemplo, si tenemos una hoja de estilos CSS compuesta de ciertos atributos con ciertos valores, pueden existir diferencias al ejecutar el mismo documento en otro navegador.

¿Hasta cuándo debería ser considerado un navegador como actual? Es una difícil pregunta, y depende del proyecto y de los requisitos del cliente. Sin embargo, veamos unos valores estadísticos respecto al uso de los navegadores en la actualidad que nos proporciona NetMarketShare.com. [24]

Como podemos observar en la siguiente ilustración el navegador más empleado en todo el mundo es Microsoft Internet Explorer con una cuota del 55,81% de usuarios.

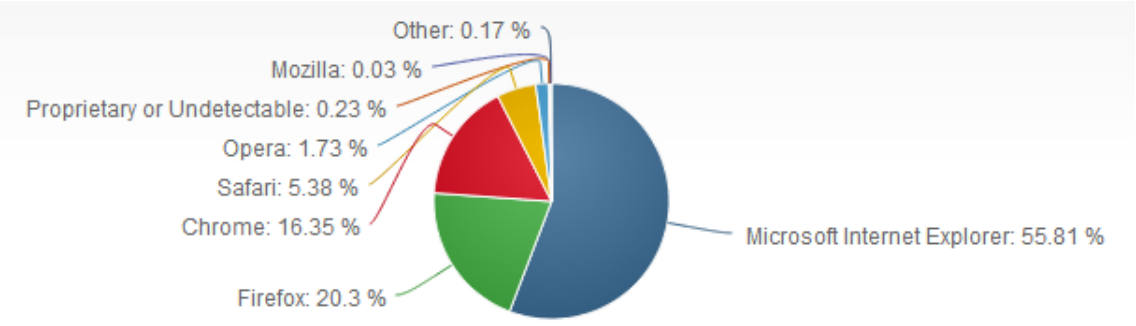


Ilustración 15 Navegadores empleados a nivel mundial

Sin embargo, existen multitud de versiones de este navegador las cuales muchas empresas y proyectos software se han decantado por dejar de dar soporte por los problemas que impactan las nuevas tecnologías. Aunque existan multitud de problemas de compatibilidades con versiones posteriores a Internet Explorer 7, vemos que la versión 6.0 aún se mantiene empleando en el mundo con una cuota del 6,22%.

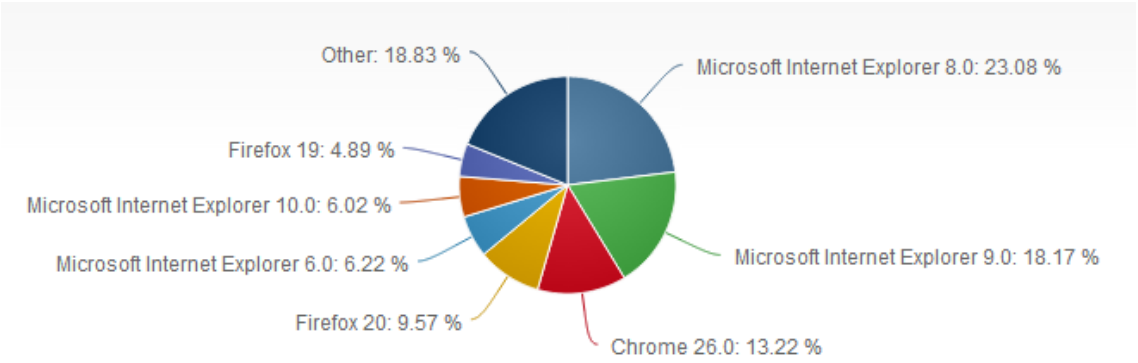


Ilustración 16 Navegadores empleados en el mundo por versiones

A continuación describiré algunas de las técnicas más famosas para conseguir que nuestras páginas webs sean cross-browsers:

Hojas de Estilos Específicas

Debido a los problemas de compatibilidades de las versiones de Internet Explorer, Microsoft dio soporte a una función para cargar hojas de estilos específicas para la versión de Internet Explorer que deseemos.

Por ejemplo, si embebemos este comentario en el código, será interpretado cargando la hoja de estilos correspondiente si el Navegador se trata de Internet Explorer 6:

```
<!--[if IE6]>
      <link href="ie.css" rel="stylesheet" type="text/css" />
<![endif]-->
```

Además, tiene un operador para determinar rango de versiones, por ejemplo con este ejemplo cargaremos la hoja de estilos correspondiente en Internet Explorer 7 y versiones posteriores.

```
<!--[if gte IE 7]>
<link href="ie.css" rel="stylesheet" />
<![endif]-->
```

Validación

Una buena opción a la hora de escribir código HTML Y CSS es seguir los estándares recomendados por el W3C, para posteriormente ser validado cada uno de ellos

a través de los validadores oficiales que nos ofrecen para HTML (http://validator.w3.org/) y CSS (http://jigsaw.w3.org/css-validator/).

Una web que es validada por el W3C será mayormente compatible con la mayoría de los navegadores, sobre todo en las últimas versiones las cuales apuntan y apoyan el compromiso del cumplimiento de los estándares.

Reset CSS

El objetivo de un Reset CSS es el de reducir las inconsistencias de interpretación de los navegadores al trabajar con hojas de estilos CSS. Las típicas diferencias por defecto que encontramos se encuentran en el espaciado entre líneas, en los márgenes, el tamaño de letra de las cabeceras, o el tamaño por defecto de la fuente. El objetivo de esta técnica es la de establecer predeterminadamente algunas de las propiedades a un valor por defecto, usualmente 0, para que se tomen como referencia para cualquier navegador.

Por defecto los navegadores emplean dos hojas de estilos, la hoja de estilo del propio navegador y las hojas de estilos del usuario. Las hojas de estilo del usuario establecen un estilo inicial de elementos Html, para atributos como el tamaño de letra, márgenes, paddings. Si no sobrescribimos ninguna de estas reglas CSS se tomarán por defecto las reglas que el navegador web tenga predeterminadas, tomando estas como prioridad máxima. El problema se encuentra en que estas reglas no sobrescritas tomarán por defecto las que cada navegador tenga establecido, difiriendo la forma de previsualización de los elementos.

La solución a este problema consiste en establecer a cero algunos estilos de elementos HTML, para posteriormente a cada clase aplicarle su valor adecuado sobrescribiendo los valores. El reset CSS más sencillo es eliminar todo los efectos de los márgenes y los padding. Por ejemplo:

```
*{
 margin: 0;
 padding: 0;
 }
```

Sin embargo, con esto no es suficiente, aunque seguramente eliminaremos bastantes problemas de maquetación de los layouts. Una solución más eficiente es neutralizar todos los estilos que puedan afectar a la forma de renderizarse en nuestro navegador. Todos estos elementos que establezcamos deben de aplicarse antes que el resto de estilos, y típicamente se incluyen en el comienzo de los documentos CSS.

Gracias a esta técnica reiniciamos todos los estilos que se aplican por defecto y empezaríamos de cero con la estilización de los elementos del HTML. Veamos un ejemplo de código que aplica esta técnica proporcionado por Meyer Web y de dominio público (http://meyerweb.com/eric/tools/css/reset/):

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
      margin: 0;
      padding: 0;
      border: 0;
      font-size: 100%;
      font: inherit;
      vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
      display: block;
}
body {
      line-height: 1;
}
ol, ul {
      list-style: none;
}
blockquote, q {
      quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
      content: '';
      content: none;
}
table {
      border-collapse: collapse;
      border-spacing: 0;
}
```

Internet Explorer y su problema de compatibilidad con los MediaQueries

Internet Explorer, como hemos visto, es un navegador web que aún mantienen muchos usuarios en sus versiones posteriores, las cuales tienen grandes problemas de compatibilidades con las nuevas tecnologías y funcionalidades que el desarrollo web posee actualmente.

Si deseas que tu Responsive Web Design se vea perfectamente en IE7 u IE8 hay que buscar otras soluciones, ya que en esas versiones Microsoft no da soporte a los media queries. Una decisión que toman muchos desarrolladores es no dar soporte a este tipo de navegadores deprecados, pero si tu cliente exige esta funcionalidad multi-navegador es un objetivo arriesgado y complicado.

Una solución para este problema es una librería externa JavaScript que simula los mediaqueries CSS. Esta librería se llama Respond (https://github.com/scottjehl/Respond) y simplemente hay que añadirla a nuestro código html dentro de la etiqueta head la siguiente línea:

```
<script type="text/javascript" src="js/respond.min.js"></script>
```

El objetivo de este script es el de proveer un versión minimizada, ocupa tan solo 3 kilobytes, bastante ligero, para dar solución a estos problemas de compatibilidad y dar soporte a navegadores obsoletos haciendo que estos sean capaces de dar soporte a los Media Queries de CSS3.

Básicamente este script inspecciona nuestras hojas de estilos CSS y a través de expresiones regulares busca en su contenido los media queries y sus bloques asociados. En Internet Explorer, el contenido de las hojas de estilos es imposible de obtener en un estado preparseado, por lo que este script emplea llamadas Ajax para parsear los CSS que recibe como respuesta. Cada bloque es concatenado a los elementos a través del atributo style y este atributo es activado o no, dependiendo de las resoluciones del navegador.

Modernizr

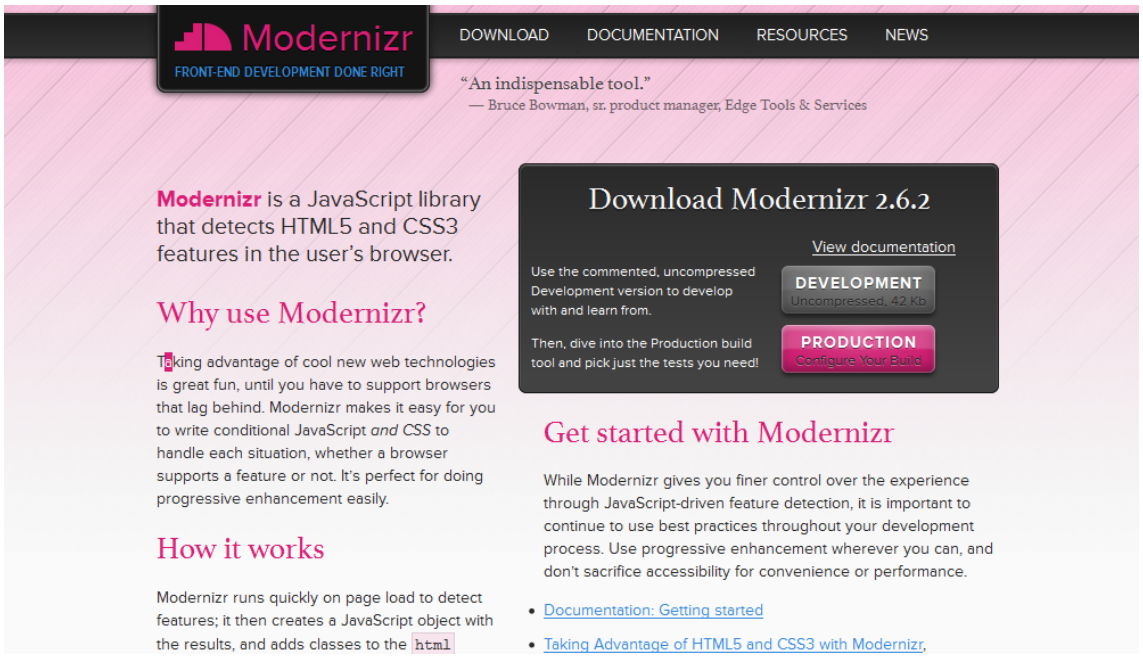


Ilustración 17 Página web de Modrnizr

Modernizr se trata de una librería javascript que detecta las funcionalidades de HTML5 y CSS3 que son compatibles con el navegador que este ejecutándose. Esta librería permite escribir Javascript y CSS condicional para manejar cada situación en caso de que el navegador sea compatible con la característica o no.

Básicamente Modernizr pregunta al navegador si soporta una funcionalidad implementada, en caso afirmativo se ejecutará normalmente. Sin embargo, si el navegador no lo soporta podremos decirle cual es el comportamiento adecuado en esa situación. La ventaja de esto en los navegadores obsoletos es el de buscar tener bajo control el desarrollo y restar preocupación de que nuestro desarrollo no funcione adecuadamente bajo cierta versión de navegador.

Para hacer funcionar esta librería solo debemos de incluirla en el head de nuestro código HTML:

```
<script type="text/javascript" src="js/modernizr.min.js"></script>
```

Veamos un ejemplo breve de cómo trabaja esta librería. Imaginemos que queremos trabajar con una funcionalidad de HTML5, por ejemplo canvas. Modernizr nos propone lo siguiente:

```
if (Modernizr.canvas)
{
      // Pintar Canvas
}
else
 {
      alert('Su navegador no soporta la funcionalidad HTML5 Canvas') ;
 }
```

En el caso de que la funcionalidad canvas sea compatible, podremos ejecutar el código javascript que deseemos para implementar nuestra funcionalidad. Sin embargo, en caso de que el navegador no sea compatible tenemos el control total de que deseamos hacer, por ejemplo, podemos mostrar una alerta avisando de la incompatibilidad de la función en ese navegador.

Herramientas

Existen multitud de herramientas que nos ayudan a probar nuestros diseños webs en multitud de navegadores para comprobar cómo se vería en un navegador determinado, sin tener que tener instalada cada una de las versiones en nuestro equipo. Veamos algunas de las opciones:

Spoon.net (https://spoon.net/browsers/)

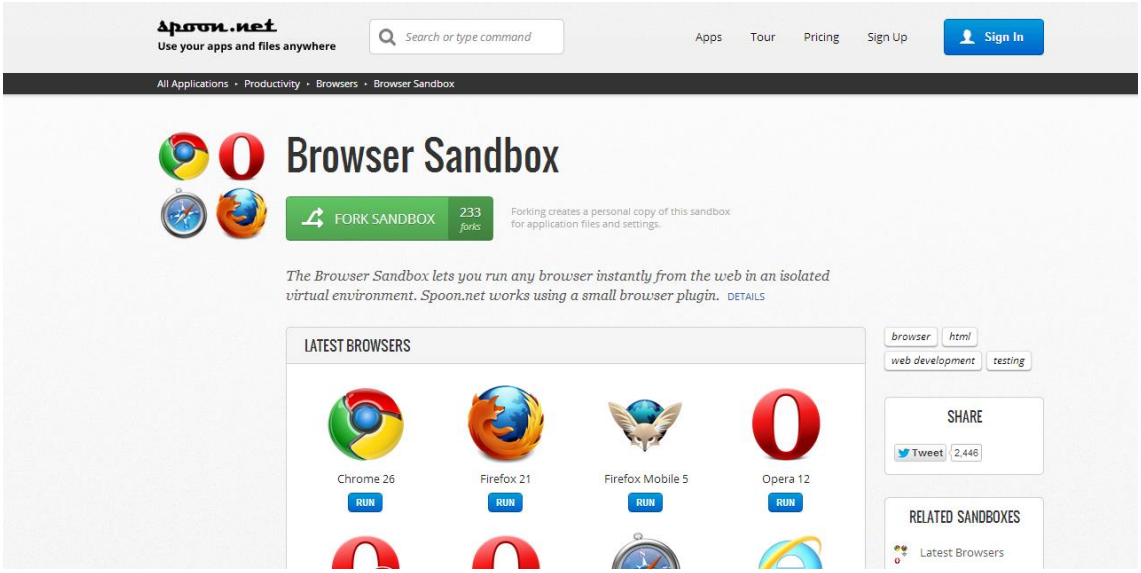


Ilustración 18 Página web de Spoon.net

Spoon es una aplicación que nos aporta un servicio de emulación de los principales navegadores para poder probar nuestro desarrollo web en cualquiera de ellos. La principal ventaja es la ejecución de las pruebas en un entorno virtual aislado, que nos evita largos procesos de instalación.

Cross Browser Testing (http://crossbrowsertesting.com/demo)

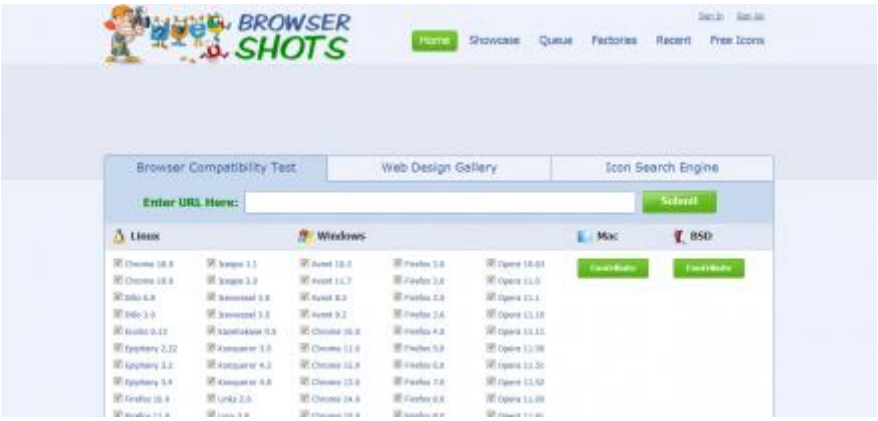


Ilustración 19 Página web de Cross Browser Testing

Browsershots se trata de un servicio online que nos permite tomar capturas de pantalla de nuestra web funcionando en distintos navegadores. La gran ventaja de esta herramienta es la gran variedad de navegadores que nos ofrece.

Net Renderer (http://netrenderer.com/)

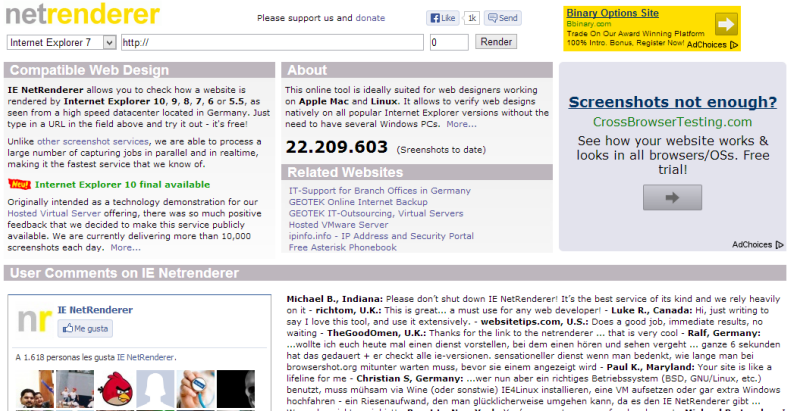


Ilustración 20 Página web de Net Renderer

Net Renderer nos permite probar la mayoría de las versiones de Internet Explorer recibiendo una captura de pantalla de cómo se visualizaría nuestra web en cada uno de ellos.

4.6 Ventajas e Inconvenientes del empleo de Responsive Web Design

4.6.1 Ventajas

El Responsive Web Design aporta distintas ventajas a la hora de implementarse. Sin embargo, estas ventajas pueden apreciarse desde distintos roles o perspectivas dentro del desarrollo web.

La primera de las ventajas, a nivel empresarial, es la reducción de costos en el desarrollo, ya que el objetivo es único y claro con una sola implementación válida tanto para dispositivos de escritorio como dispositivos móviles. La creación de webs específicas para dispositivos aumenta el coste de desarrollo. Sin embargo, un solo desarrollo de un portal web accesible desde cualquier dispositivo es bastante más barato.

Por ejemplo, como no es necesario alojar distintas versiones de la web no es necesario ser propietario de distintos dominios o el coste adicional de desarrollar una aplicación móvil.

Otra ventaja que aporta es la gran eficiencia a la hora de realizar los procesos de mantenimiento y actualización del contenido, ya que con un solo cambio se podrá ver reflejado en todos los dispositivos. Sin embargo, si poseemos distintos desarrollos para cada plataforma se deberá realizar este proceso pudiendo provocar posibilidades de error e inconsistencia además de un consumo mayor de recursos.

Por ejemplo, si deseamos realizar un cambio en la identidad corporativa del negocio y desear cambiar el color principal de la página web, con Responsive Web Design solo sería suficiente con modificar un CSS, mientras que con distintas versiones deberíamos modificar todas las hojas de estilos todas las webs o aplicación móvil.

Lo que todos los estudios reflejan es la mejora de la usabilidad, ya que la navegación por parte del usuario es mucho más cómoda, adaptándose la web al medio correspondiente facilitando la navegación. Además a nivel de consumo por parte del cliente provoca un gran impacto ya que se suele asociar el desarrollo con grandes niveles de creatividad e innovación tecnológica.

Otras de las ventajas dejando aparte la usabilidad y la interacción, se trata del posicionamiento web. Este tipo de desarrollos ayudan al posicionamiento de la web en los rankings de buscadores debido a sus características. Una de ellas se trata de poseer una única URL, ya que cuando se fragmentaron los desarrollos para diversos dispositivos se emplearon distintos dominios o subdominios para la versión correspondiente, no estando los desarrollos alineados a nivel de SEO.

Es decir, donde antes empleábamos unas URL parecidas a estas:

- http://mobile.mipaginaweb.com

Ahora podemos emplear una sola URL para todos los soportes (http://mipaginaweb.com) que nos aporta la centralización de todo el tráfico para poder manejarlo de manera eficiente, creando informes reales del tráfico producido obteniendo una visión completa de todo el tráfico (por ejemplo con Google Analytics) y acumulando todos los enlaces o pageranks. Además todas las acciones que sean necesarias que desee tomar el SEO para posicionar la web serán mucho más productivas si las desarrolla sobre una sola web que si lo tiene que hacer con varios.

Además otro ejemplo, es que antes cuando un usuario accedía a una web a través de un dispositivo móvil y compartía una URL esta llegaba a una serie de usuarios que no obligatoriamente deben de estar en un medio móvil. Para solucionar este problema era necesario crear redirecciones que dirijan a los clientes hacia el sitio web adecuado respecto del dispositivo conectado para no perder la experiencia de usuario.

Ya que con Responsive Web Design todos los dispositivos acceden a una misma URL, ya no son necesarias las redirecciones.

Además el acceso a la web desde un smartphone nos permite como desarrolladores web sacar beneficio de las tecnologías móviles explotando funcionalidades como la geo localización y los eventos táctiles.

4.6.2 Desventajas

Como todas las tecnologías no implica que Responsive Web Design sea la panacea, también sus ventajas pueden producir ciertas desventajas que debemos de tener en cuenta a la hora de implementar esta filosofía.

Uno de los problemas que puede repercutir en un diseño Responsive son los tiempos de carga, ya que el usuario debe descargar todo el código fuente, aún sin ser necesario, e incluso si el diseño no está optimizado con imágenes adaptadas a distintas resoluciones, debe descargar imágenes de gran resolución sin ser necesario.

Aunque se han comentado las ventajas SEO, no todo pinta tan bien, ya que las descripciones en títulos, imágenes y contenidos son comunes tanto para usuarios de dispositivos móviles como no, siendo distinto el objetivo de las palabras claves en búsquedas móviles como búsquedas de escritorios.

Aunque aporta ventajas para el desarrollador web, también este debe de emplear más tiempo en optimizar el diseño para las distintas resoluciones respecto de un desarrollo web no Responsive. Además de esta desventaja, todo el mundo escucha el

término y quiere transformar su web en Responsive. Sin embargo, es mucho más fácil comenzar con la filosofía que tratar de convertir un sitio web en Responsive.

4.7 Técnicas de Implementación

A continuación se van a describir las técnicas mayormente empleadas por la comunidad web a la hora de implementar un diseño Responsive diseñadas por Luke Wroblewski [25]. Estas técnicas son vocabulario básico a la hora de implementar este tipo de filosofía y ayuda a crear un layout básico sensible a la multitud de variables tamaños de pantalla.

4.7.1 Mostly Fluid (Mayormente fluido)

Esta técnica es el patrón más popular y dentro de la comunidad Responsive uno de los mayormente empleados en los distintos diseños. En dimensiones de smartphone se apilan todas las columnas y según aumentan estas dimensiones se voltean en un desarrollo multi-columna tanto como el ancho del navegador deje acomodarlas. Estos elementos se vuelven fluidos y se expanden hasta los diferentes breakpoints. Además se puede combinar con la técnica de Column Drop que veremos más adelante.

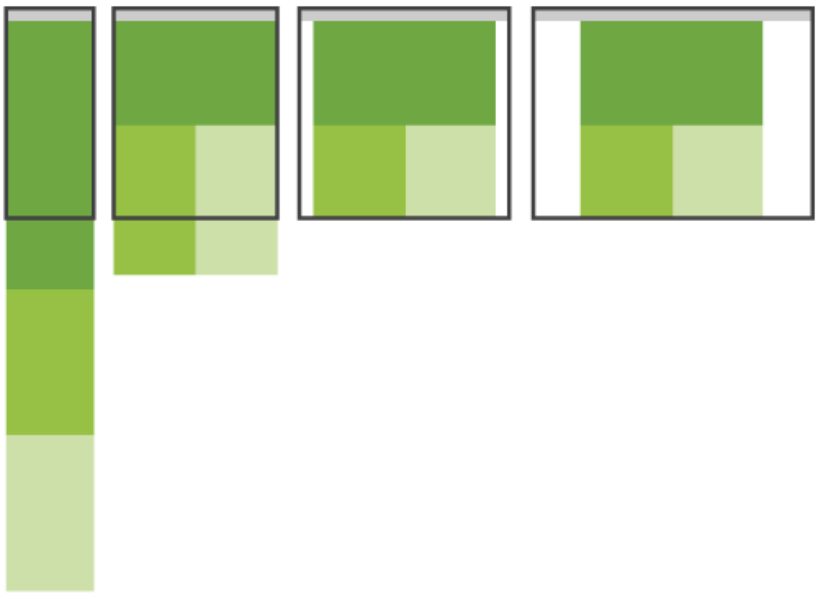


Ilustración 21 Wireframe de Mostly Fluid

Otro concepto que este patrón emplea es definir un ancho máximo para el wrapper que contiene las capas pudiendo crecer el diseño y expandirse a cualquier resolución del navegador quedando un espacio en blanco a los lados del contenido. A estas dimensiones el diseño se basa en los layouts de ancho fijo y centrados típicos de los últimos años.

El origen del nombre "mayormente líquido" es debido a que la estructura principal de los layouts no cambia realmente hasta dimensiones móvil donde se aplica la caída de columna. Como hemos comprobado las columnas se vuelven liquidas con su contenido hasta alcanzar el ancho máximo.

A continuación mostraré algunos ejemplos de webs empleando este patrón:

Five Simple Steps (http://mediaqueri.es/fss/)

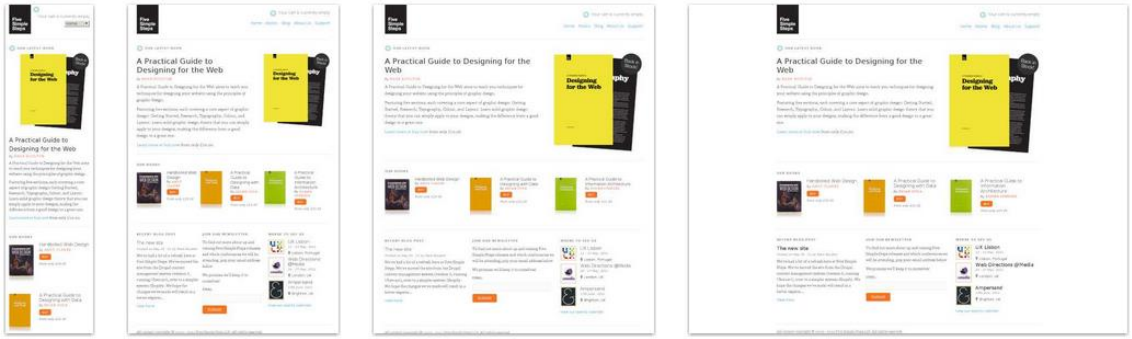


Ilustración 22 Five Simple Steps

Princess Elisabeth Antarctica (http://mediaqueri.es/pea/)

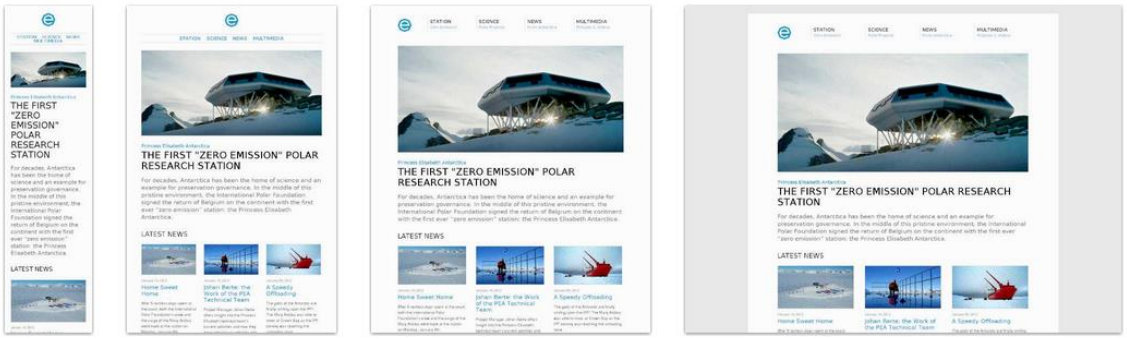


Ilustración 23 Princess Elisabeth Antarctica

Trent Walton (http://mediaqueri.es/tre/)

Ilustración 24 Trent Walton

ChoiceResponse (http://mediaqueri.es/cho/)

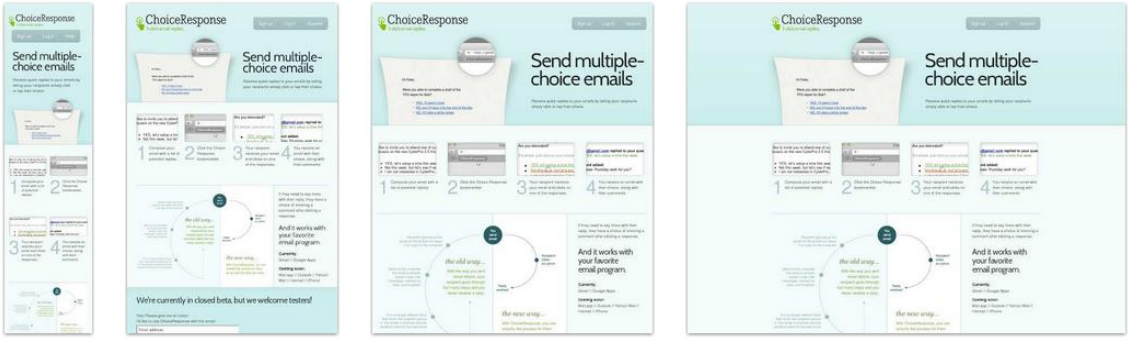


Ilustración 25 Choice Response

En estos diseño podemos comprobar como en resoluciones grandes hasta tamaño tablet el contenido fluye dimensionando el tamaño de los elementos hasta llegar a versión móvil donde los elementos se colocan de forma apilada. En cada salto cada diseño puede tener pequeños cambios, quizás de navegación o de pequeña organización, pero el layout principal marca la adaptación de todo el contenido al ancho total del navegador.

4.7.1.1 Ejemplo de Implementación

Veamos a continuación un pequeño ejemplo de implementación de esta técnica. Lo primero que vamos a definir son los elementos que marcan la estructura del HTML:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8"/>
		<meta name="viewport" content="widht=device-width"/>
		<title> Mostly Fluid</title>
		<link rel="stylesheet" href="style.css"/>
	</head>
	<body>
		<header>Header</header>
		<div id="wrapper">
			<section>
				<article id="column1">Column 1</article>
				<article id="column2">Column 2</article>
				<article id="column3">Column 3</article>
			</section>
			<footer></footer>
		</div>
	</body>
</html>
```

Definimos una cabecera con la etiqueta <header> y al mismo nivel una capa con el identificador wrapper cuya función será la de establecer la envoltura de nuestros elementos. Tras esto crearemos una sección donde se encontrarán 3 capas que organizarán la distribución de nuestros elementos.

Para empezar definimos los valores de nuestro wrapper. Primeramente esta capa irá centrada a través de la opción margin: 0 auto que ajustará la capa sin margen superior y con los márgenes laterales manteniendo la capa siempre en el centro. El ancho definido será de un 80%, dejando un tamaño del 10% a cada lado libre.

```
#wrapper
{
	width: 80%;
	margin: 0 auto;
}
```

Cada uno de los elementos definidos pertenece a article, por lo que estableceremos unos atributos para todos ellos que mantendrán en común:

```
article

{
	-webkit-box-sizing: border-box;
	-moz-box-sizing: border-box;
	box-sizing: border-box;
	float:left;
	height: 100%;
	background-color: grey;
	padding: 2%;
	text-align: center;
	height: 400px;
}
```

En el código observamos como empleamos la técnica de box-sizing comentada para facilitarnos los cálculos matemáticos de las capas. Además todas las capas estarán flotadas a la izquierda.

El primer elemento se tratará de una capa que ocupará el 100% de nuestro wrapper. Este 100% de ancho hace referencia al elemento inmediatamente superior, que en nuestro caso será la capa wrapper. Al ocupar el 100% no dejará espacio para ningún elemento más.

```
#column1
{
background-color: #96c144;
width: 100%;
}
```

Inmediatamente debajo nos encontramos con la column2 y la column3, las cuales se van a repartir el espacio al 50%.

```
#column2
{
background-color: #6fa843;
width: 50%;
}

#column3
{
background-color: #cde0aa;
width: 50%;
}
```

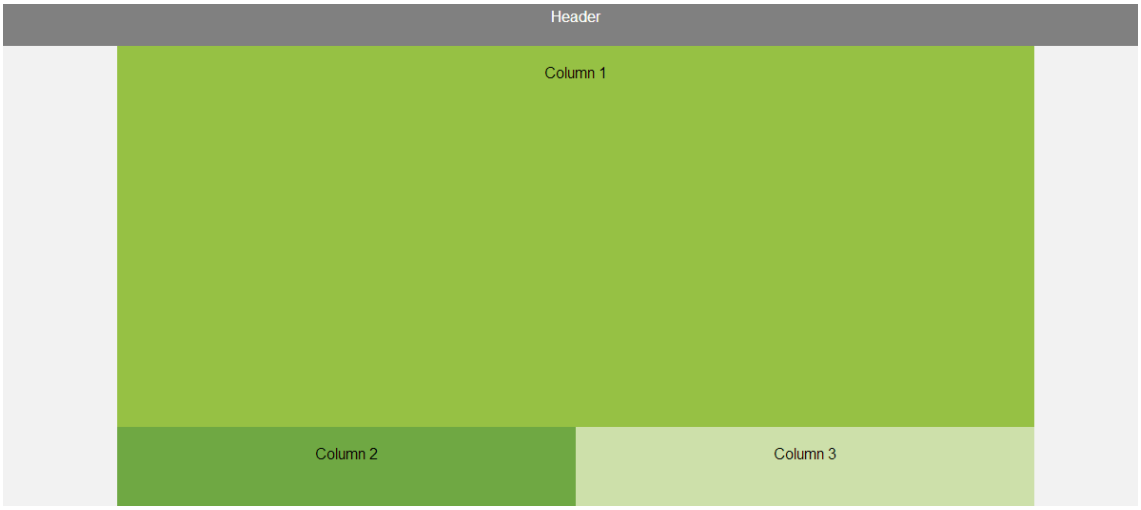


Ilustración 26 Ejemplo Mostly Fluid 1

En este instante aunque redimensionemos el navegador, al tratar los anchos con porcentajes, siempre se mantendrá esta estructura, pero como queremos reorganizar los elementos a menores tamaños veremos cómo jugamos con los mediaqueries.

La primera decisión de diseño es que al llegar a un ancho de 960 pixeles, vamos a aumentar el espacio del wrapper a un 90% dejando un espacio del 5% a cada lado:

```
@media screen and (max-width: 960px){

	#wrapper
	{
		width: 90%;
		margin: 0 auto;
	}
}
```

Ilustración 27 Ejemplo Mostly Fluid 2

Se mantiene el diseño manteniendo unos márgenes más pequeños a los lados, hasta que llegamos a un ancho de 640 pixeles. En este instante la decisión de diseño es eliminar estos espacios molestos en tablets, haciendo que todo el wrapper ocupe el 100% del ancho.

```
@media screen and (max-width: 640px){

	#wrapper
	{
		width: 100%;
		margin: 0 auto;
	}
}
```

Ilustración 28 Ejemplo Mostly Fluid 3

Como vemos, hasta ahora todo ha seguido la técnica de mayormente fluido, manteniendo la organización de los elementos pero reduciendo su ancho proporcionalmente al de la página. Finalmente en la versión para móvil no interesa mantener esta estructura por que los elementos se reducirían tanto que no se apreciarían y daríamos una mala experiencia al usuario. Por lo consiguiente haremos ocupar el 100% del ancho a cada elemento para que queden totalmente apilados:

```
@media screen and (max-width: 480px){


#wrapper
{
      width: 100%;
      margin: 0 auto;
}

#column1
{
      width: 100%;
}

#column2
{
      width: 100%;
}

#column3
      {
      width: 100%;
      }
}
```

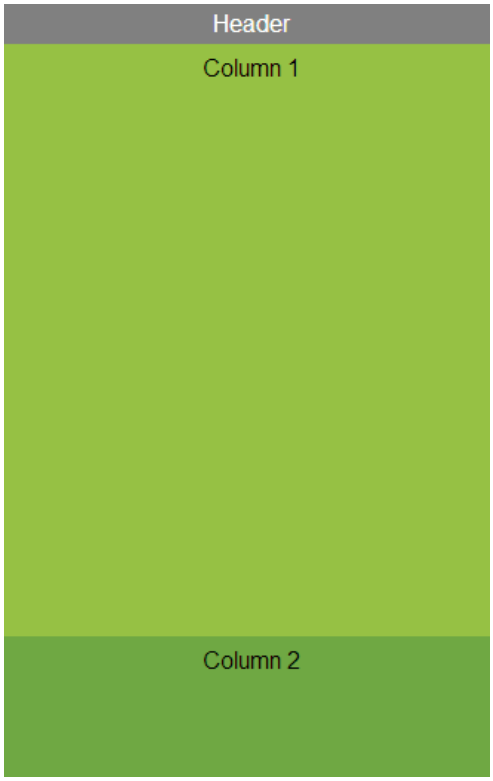


Ilustración 29 Ejemplo Mostly Fluid 4

4.7.2 Column Drop (Caída de columnas)

La siguiente técnica a describir se denomina Column Drop o en español Caída de Columna. Es un patrón popular que comienza con un layout multicolumna y termina con un layout sencillo de una columna, dejando caer las columnas a medida que la dimensión de la pantalla se va estrechando.

A diferencia del patrón Mostly Fluid, el tamaño total de los elementos en esta disposición tiende a permanecer constante. En este diseño la adaptación a distintos tamaños de pantalla se basa en apilar las columnas como en el siguiente wireframe:

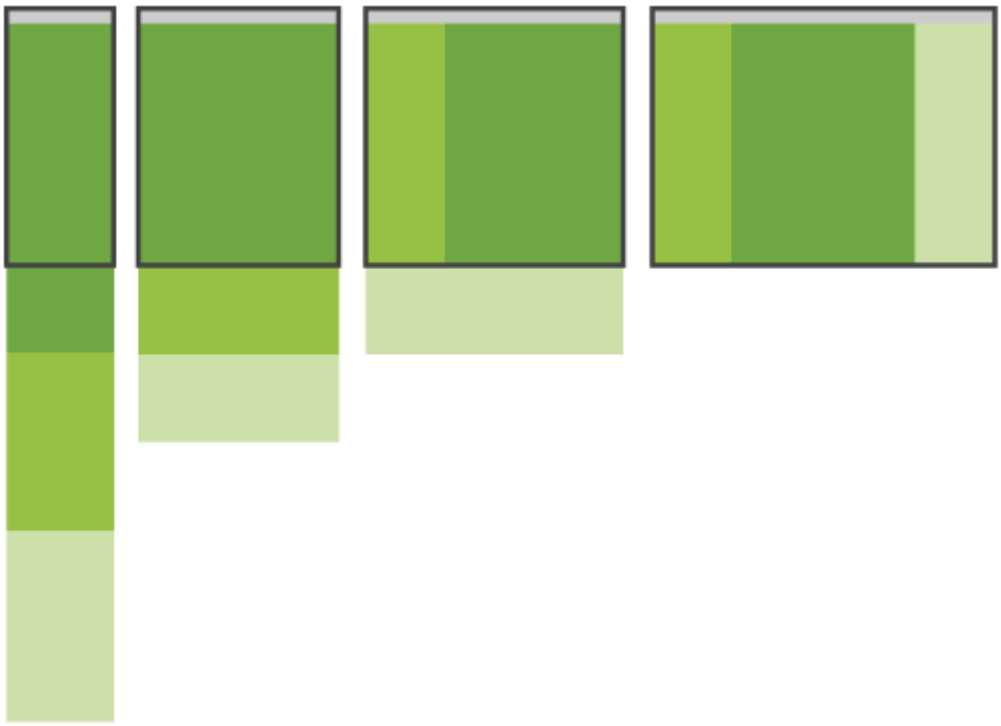


Ilustración 30 Wireframe de Column Drop

Saber el cómo y cuándo cada columna se debe de apilar en los breakpoints de resolución no es un dogma, debe ser es diferente para cada diseño. A continuación mostraré algunos ejemplos de webs que emplean este patrón. [26]

Modernizr (http://mediaqueri.es/mod/)

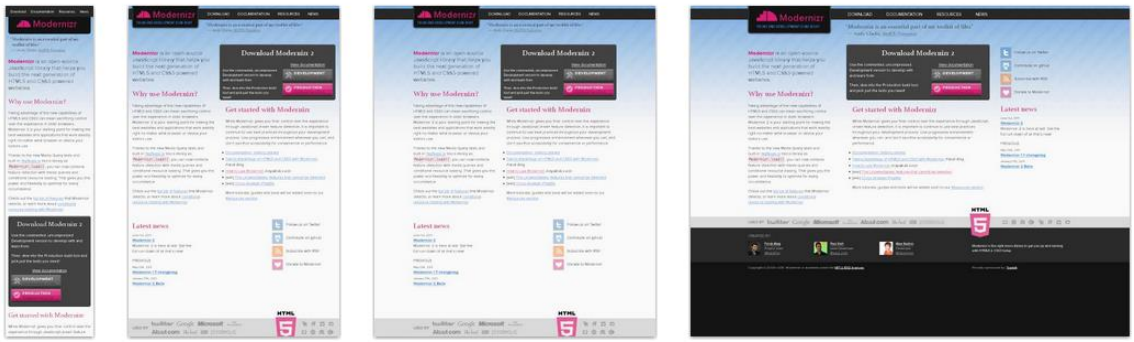


Ilustración 31 Modernizr

Wee Nudge (http://mediaqueri.es/wee/)

Ilustración 32 Wee Nudge

Ash Personal Training (http://mediaqueri.es/ash/)

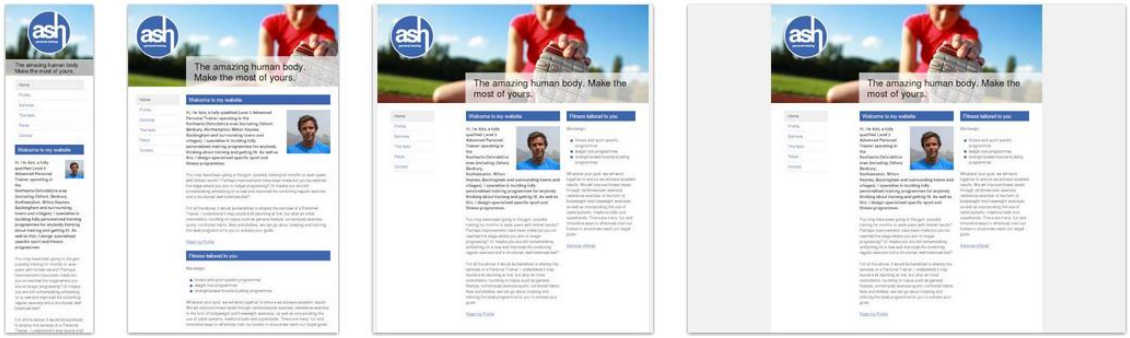


Ilustración 33 Ash Personal Training

Festival de Saintes (http://mediaqueri.es/fds/)

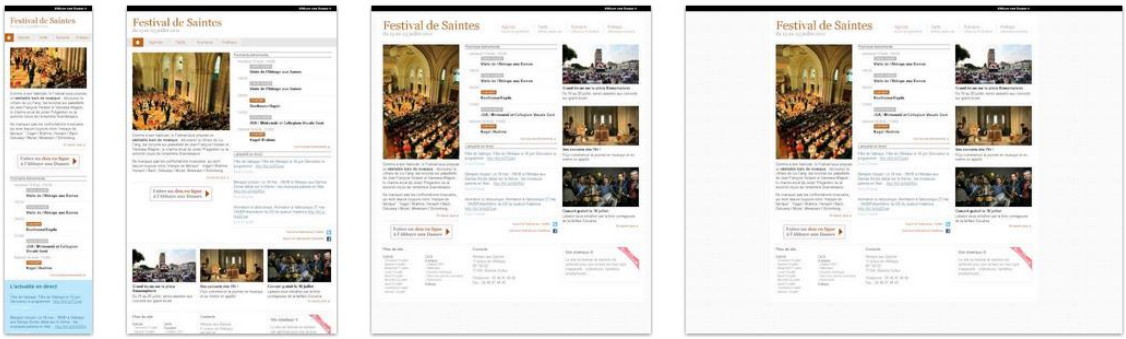


Ilustración 34 Festival de Saintes

Como vemos todos estos diseños se tratan de diseños basados en columnas, al llegar a un ancho determinado una de las columnas, generalmente la columna de la derecha, se deja caer a la parte inferior del diseño ocupando el 100%. Esta caída de

columnas está determinada por el número de columnas y el momento en el que el diseño pida espacio para una buena usabilidad.

4.7.2.1 Ejemplo de Implementación

Veamos a continuación un pequeño ejemplo de implementación de esta técnica. Lo primero que vamos a definir son los elementos que marcan la estructura del HTML, siendo la estructura completamente igual que en la técnica anterior:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8"/>
		<meta name="viewport" content="widht=device-width"/>
		<title> Column Drop</title>
		<link rel="stylesheet" href="style.css"/>
	</head>
	<body>
		<div id="wrapper">
			<header>Header</header>
			<section>
				<article id="column1">Column 1</article>
				<article id="column2">Column 2</article>
				<article id="column3">Column 3</article>
			</section>
			<footer></footer>
		</div>
	</body>
</html>
```

Como se trata de un layout basado en columnas repartiremos el 100% del ancho disponible en dos columnas del 20% y en una central del 60%:

```
#column1
{
	background-color: #96c144;
	width: 20%;
}

#column2
{
	background-color: #6fa843;
	width: 60%;
}
#column3
{
	background-color: #cde0aa;
	width: 20%;
}
```

La estructura queda de esta manera:

Ilustración 35 Ejemplo Column Drop 1

A continuación tras estrechar la pantalla debemos de proceder a realizar la caída de columna. De tal manera las dos primeras columnas pasaran a repartirse el espacio disponible y la tercera columna caerá justo debajo ocupando el 100%. Como vemos del 100% la columna 1 y la columna 2 se reparten un 34% y un 66% respectivamente:

```
@media screen and (max-width: 800px){

	#column1
	{
		width: 34%;
	}

	#column2
	{
		width: 66%;
	}

	#column3
	{
		width: 100%;
		height: 200px;
	}
}
```

El resultado queda así:

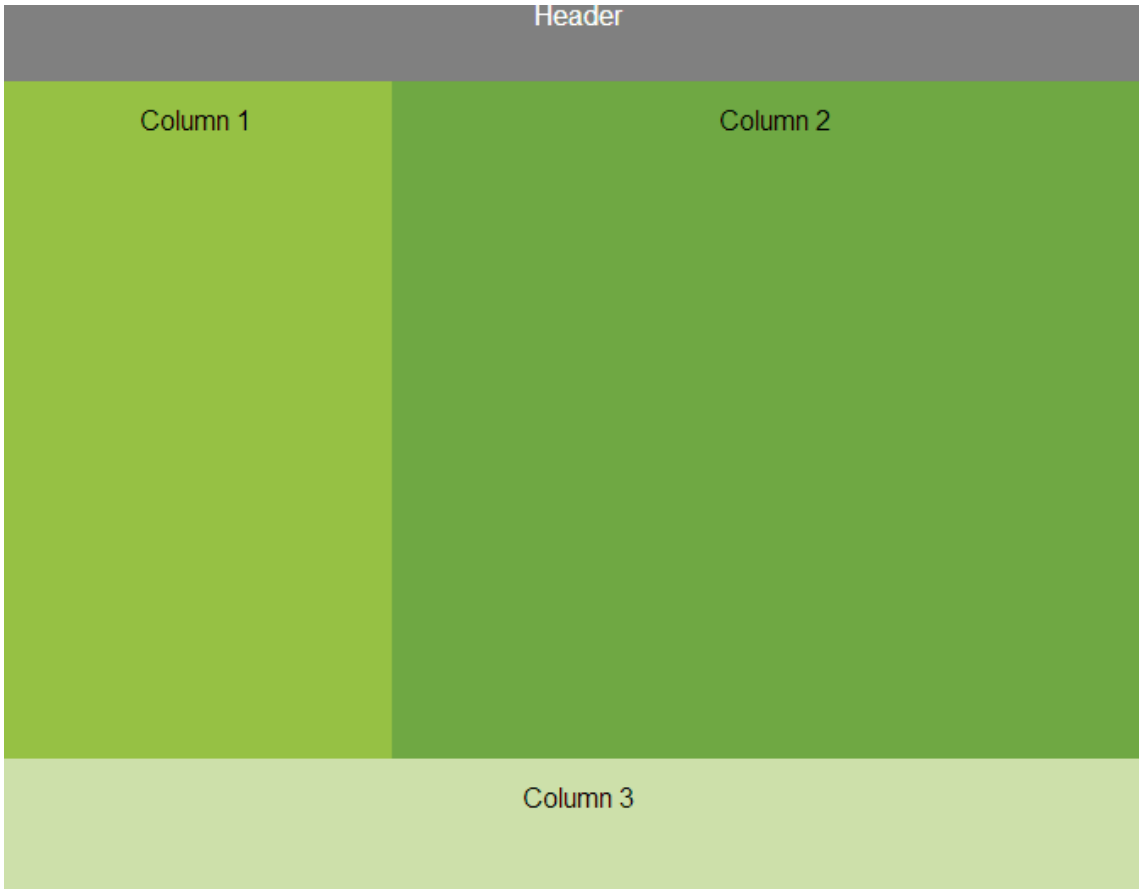


Ilustración 36 Ejemplo Column Drop 2

Finalmente tras estrechar más el diseño la decisión que se toma es que caiga la segunda columna quedando las tres columnas iniciales apiladas. Simplemente debemos de asignar el 100% del espacio a cada una de las columnas:

```
@media screen and (max-width: 480px){

        #column1
        {
        width: 100%;
        }
        #column2
        {
        width: 100%;
        }
        #column3
        {
        width: 100%;
        }

        header
        {
        width:100%;
        }
}
```

Quedando como resultado:

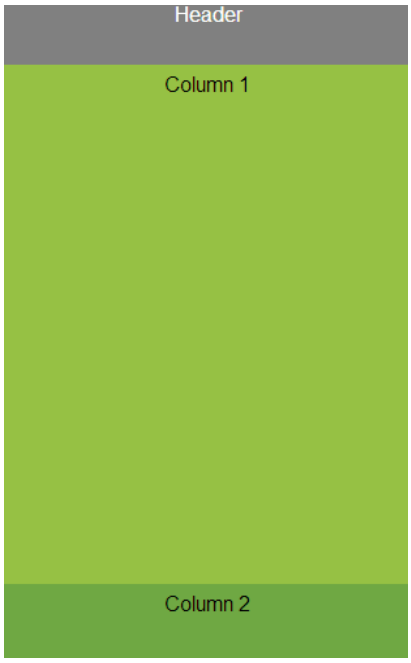


Ilustración 37 Ejemplo Column Drop 2

4.7.3 Layout Shifter (Cambio de layout)

Este patrón es menos popular que los anteriores, aunque por otro lado se consigue adaptarse bastante bien a los tamaños de pantallas ya que se aplican diferentes diseños según el tamaño de las pantallas. Esto evidentemente implica bastante más trabajo para ajustar todos los elementos a las diferentes resoluciones y saltos en contraposición a los layouts anteriores que eran más fluidos.

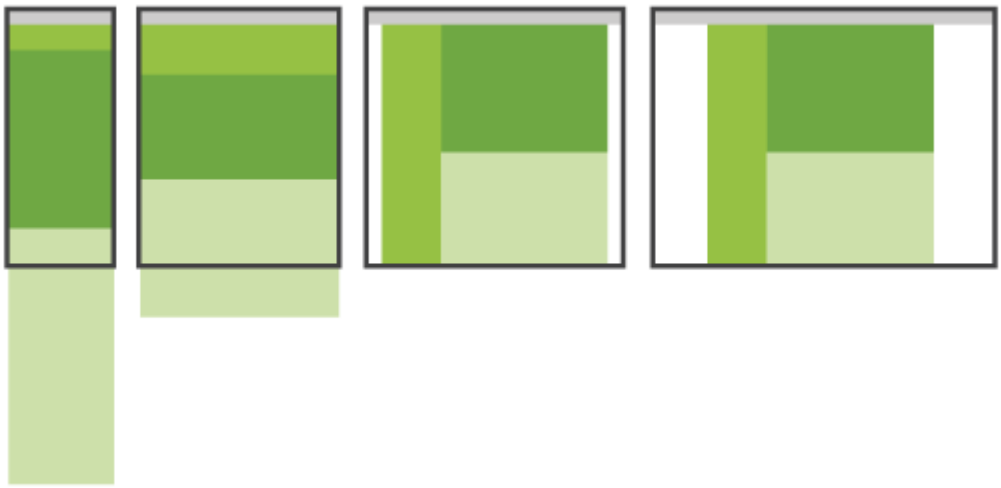


Ilustración 38 Wireframe de Layout Shifter

Este layout es complicado de generalizar en un tipo de estructura, ya que es el más innovador y cada salto puede emplear otras de las técnicas descritas anteriormente. Algunos diseños que emplean esta técnica pueden ser los siguientes:

Food Sense (http://mediaqueri.es/fse/)

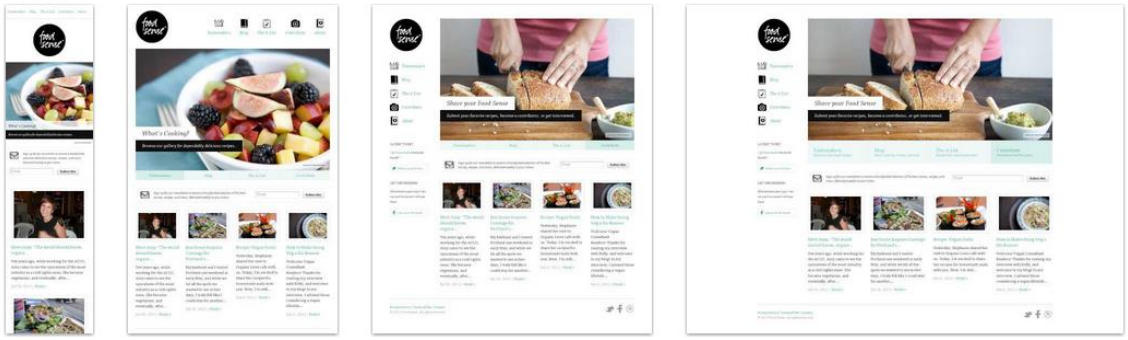


Ilustración 39 Food Sense

The Boston Globe (http://mediaqueri.es/bg/)

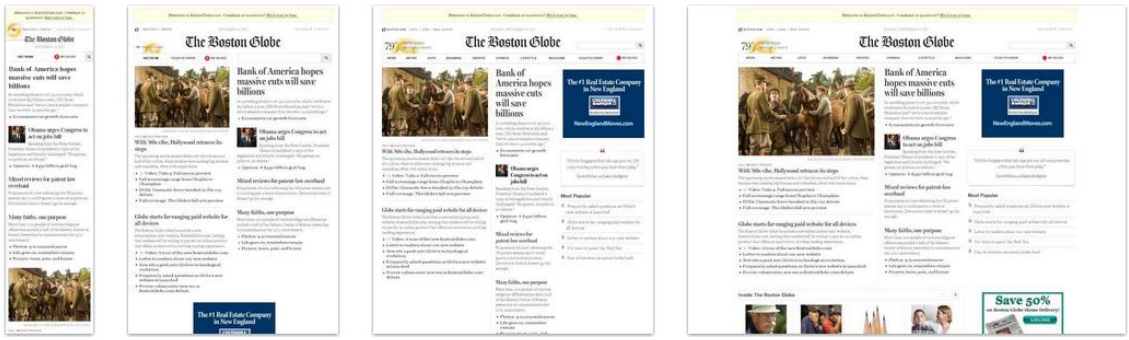


Ilustración 40 The Boston Globe

Andersson Wise Architects (http://mediaqueri.es/awa/)

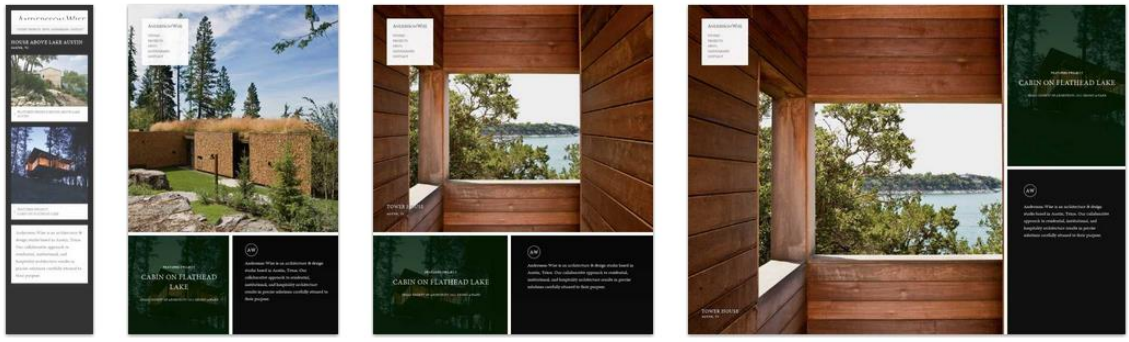


Ilustración 41 Andersson Wise Architects

Como vemos en estos distintos diseños mezclan las dos técnicas explicadas anteriormente. En algunos saltos se toma la decisión de caídas de columnas mientras en otros se mantiene la organización pero fluyendo el contenido proporcionalmente.

4.7.4 Tiny Tweaks (Cambios Minúsculos)

Este patrón se puede definir como la implementación más sencilla y minimalista. Este layout desprende simplicidad, adaptación multidispositivo y dar la sensación de adaptación a través del tamaño de fuentes e imágenes. Como vemos en el wireframe se trata de una capa principal que reduce los elementos interiores. Se debe de emplear para layouts simples sin alto contenido de elementos y menús, ideales para landing pages:

Ilustración 42 Wireframe Tiny Tweaks

A continuación veamos unos ejemplos de diseños que emplean esta técnica:

Future Friendly (http://mediaqueri.es/ff/)

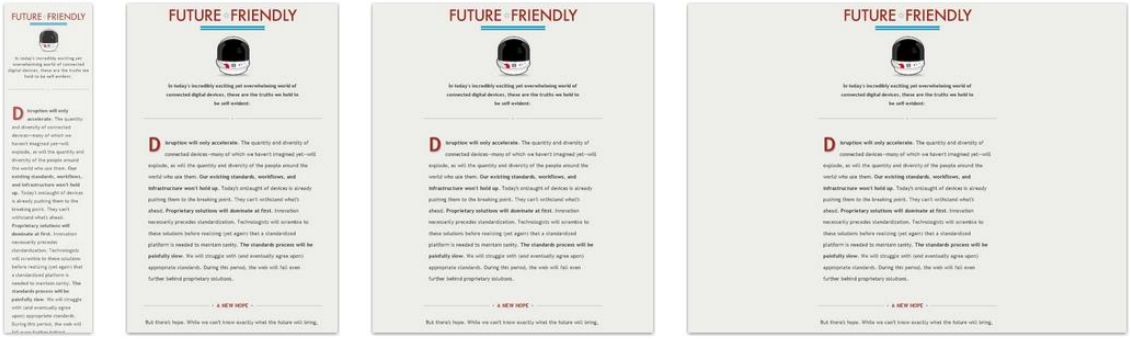


Ilustración 43 Future Friendly

Neovada (http://mediaqueri.es/ff/)

Ilustración 44 Neovada

Lycos (http://mediaqueri.es/lyc/)

Ilustración 45 Lycos

Como vemos en estos diseños existe una capa principal que es la que mantiene el tamaño de todo el contenido de la página disminuyendo sus elementos pero sin apenas una reorganización de éstos. Se juega más con los tamaños de fuentes e imágenes.

4.7.5 Off Canvas (Fuera del Lienzo)

Para terminar con las distintas secciones de layouts, conozcamos el patrón Off Canvas. La idea de esta técnica es aplicarla en resoluciones pequeñas, por ejemplo smartphones, para aprovechar todo el espacio disponible y crear una navegación cómoda para el usuario. Como hemos visto todos los diseños anteriores al final en pantallas pequeñas tienden a apilarse los elementos verticalmente.

Ésta técnica intenta acercarse a los diseño de apps para móvil, ya que el objetivo es ocultar una capa fuera del espacio visible, aprovechando este espacio para mantener más contenido o en la mayoría de los casos la navegación, ocultándose en dimensiones donde no es realmente cómodo ir apilando capas lo que crea largas páginas verticales.

Como vemos en el Wireframe en tamaños grandes puede poseer distinto diseño, pero al llegar a dimensiones del viewport donde el espacio es importante, se sacan del lienzo y se superponen en la parte visible a través de alguna interacción de usuario. Para ello debemos de ayudarnos de JavaScript.

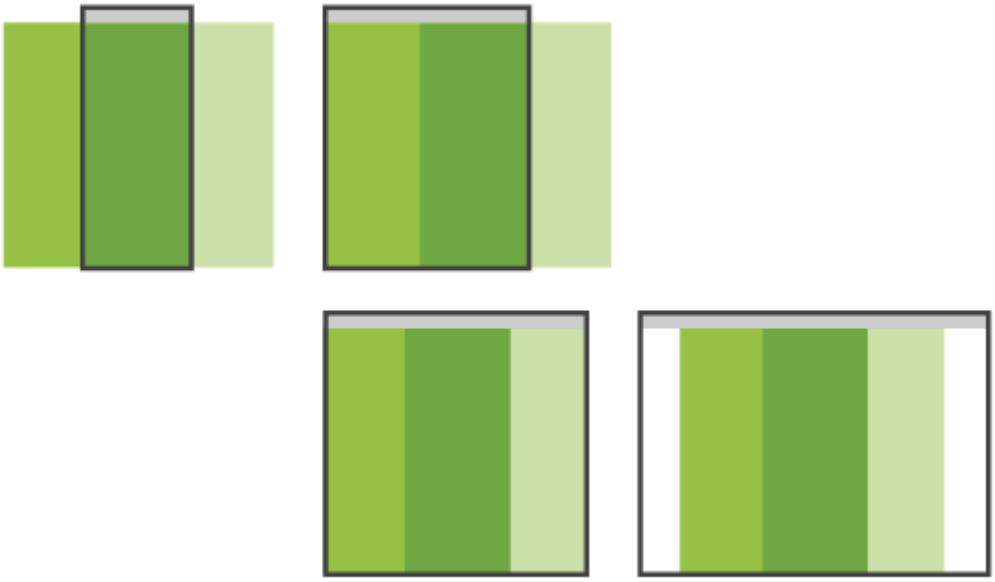


Ilustración 46 Wireframe OffCanvas

En muchas aplicaciones web, es común emplear un espacio a un lado de la pantalla, donde se ocultan opciones de navegación hasta que se interactúa con un elemento que hace que el contenido de la pantalla se deslice y se superponga esta capa. Un ejemplo común es la app móvil html5 de Facebook.

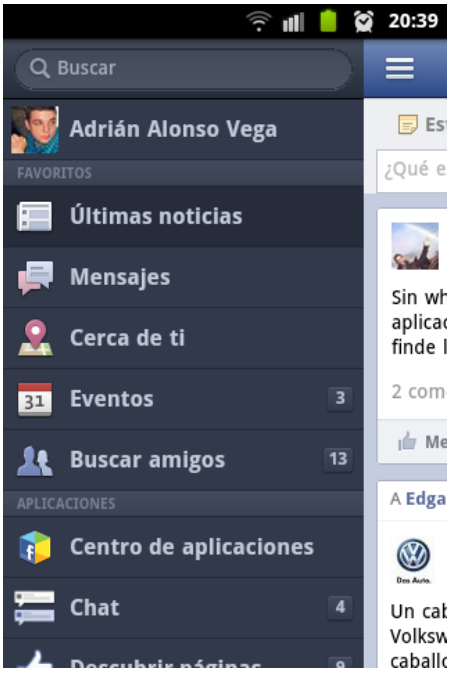


Ilustración 47 Aplicacion HTML5 de Facebook

Además se puede aprovechar no solo para tamaño smartphones también para aplicaciones web cuando la dimensión de la pantalla es de una Tablet. Este patrón aporta originalidad, ahorro de espacio y sobre todo innovación.

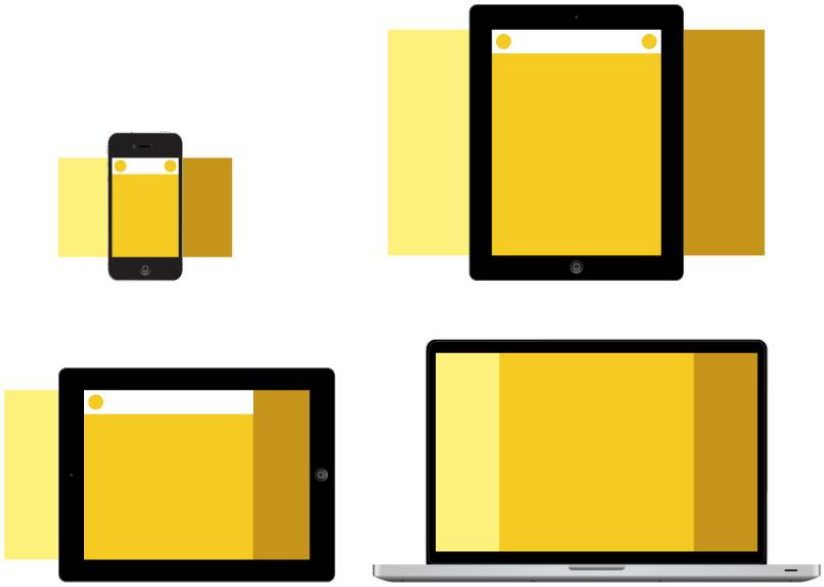


Ilustración 48 Wireframes del patron Off Cavnas (fuente: http://www.elmastudio.de/webdesign/responsive-webdesign-2/off-canvas-layouts-in-responsive-webdesigns/)

A continuación veamos algunos ejemplos que actualmente emplean este tipo de técnica en diseños Responsive Web Design:

Kaemingk (http://www.kaemingk.com/nl/)

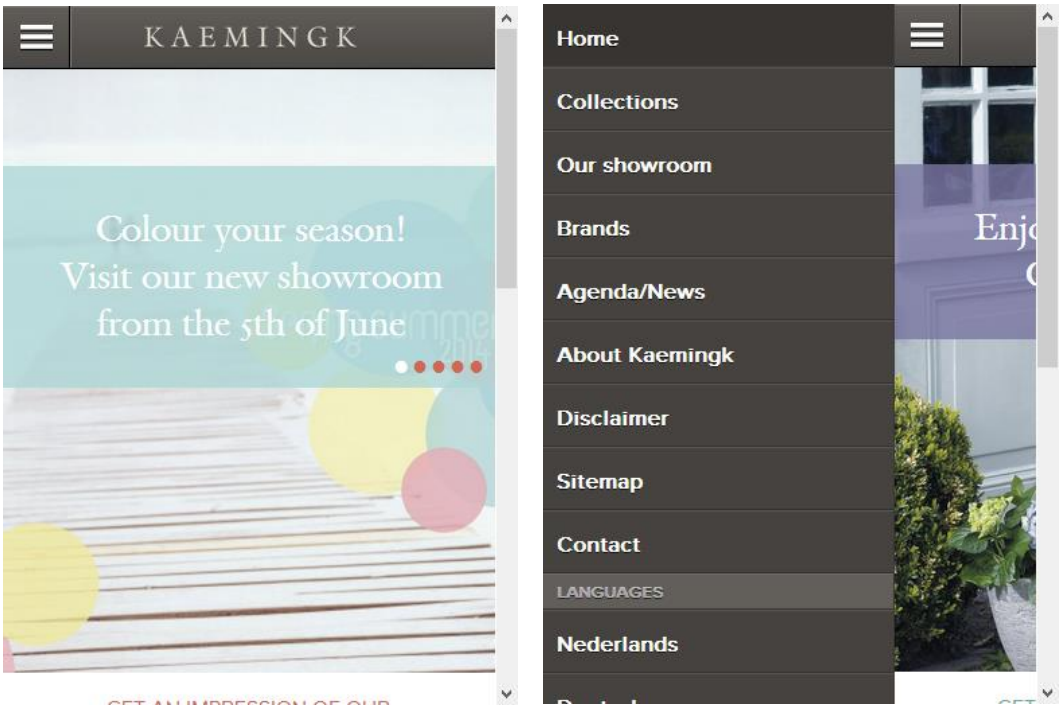


Ilustración 49 Web de Kaemingk mostrando la técnica de Off Canvas

4.8 Herramientas para probar diseños Responsive Web Design

Durante el proceso de diseñar nuestra página web Responsive Web Design, la tarea más importante es la de testear nuestro diseño. Es un proceso constante durante todo el proceso de desarrollo. La manera más fácil de probar es redimensionando manualmente nuestro navegador así viendo el resultado de la fluidez al reducir el ancho del dispositivo. Sin embargo, la forma óptima sería probarlo en dispositivos físicos.

En general los clientes quieren que el diseño se adapte perfectamente a terminales concretos como pueden ser iPhone e Ipad. Para ayudar a este problema existen multitud de herramientas que nos ayudan recorrer distintos tamaños de pantalla simulando un dispositivo para que podamos probar nuestro diseño a distintos dispositivos, pantallas y orientaciones.

A continuación presentaré herramientas online que nos ayudan a probar nuestro diseño en distintas resoluciones:

Quirktools.com / Screenfly (https://quirktools.com/screenfly/)

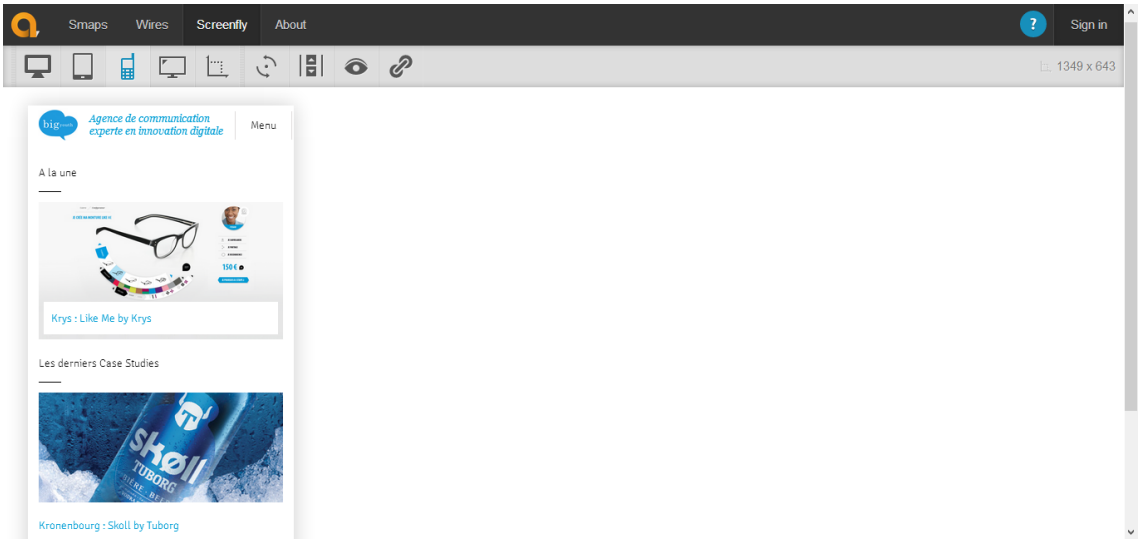
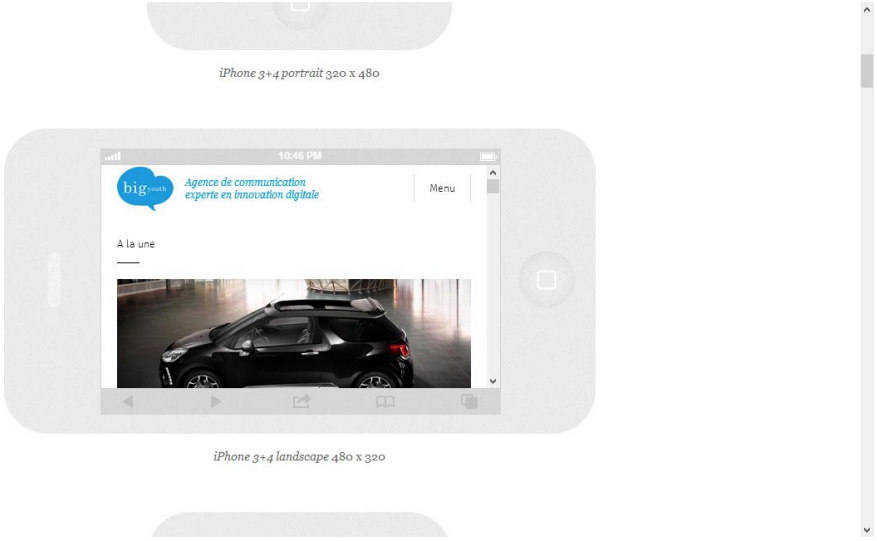


Ilustración 50 Página web de ScreenFly

ScreenFly proporciona este conjunto de herramientas, por ejemplo nos permite redimensionar con resoluciones de terminales conocidos en el mercado. También nos permite cambiar la orientación del dispositivo, activar el scroll o ajustar a las dimensiones que nosotros deseemos.

Responsinator (http://www.responsinator.com/)



Responsinator nos amolda nuestro diseño proporcionándonos como se visualizarían en dispositivos conocidos como IPhone, Ipad o dispositivos Android.

Ilustración 51 Página web de Responsinator

Resize my Browser (http://resizemybrowser.com/)

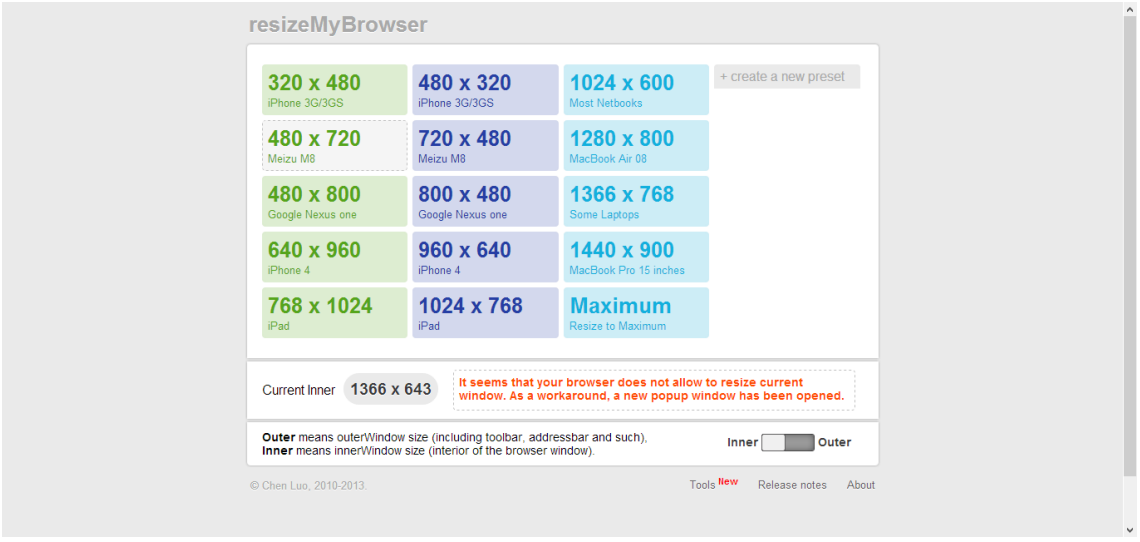


Ilustración 52 Página web de Resize My Browser

Resize MyBroswser es una herramienta online que nos ofrece resoluciones conocidas para ciertos dispositivos haciendo que se abra una nueva ventana del navegador con el tamaño seleccionado para probar en esas dimensiones la url que deseemos.

Screen Queries (http://screenqueri.es/)

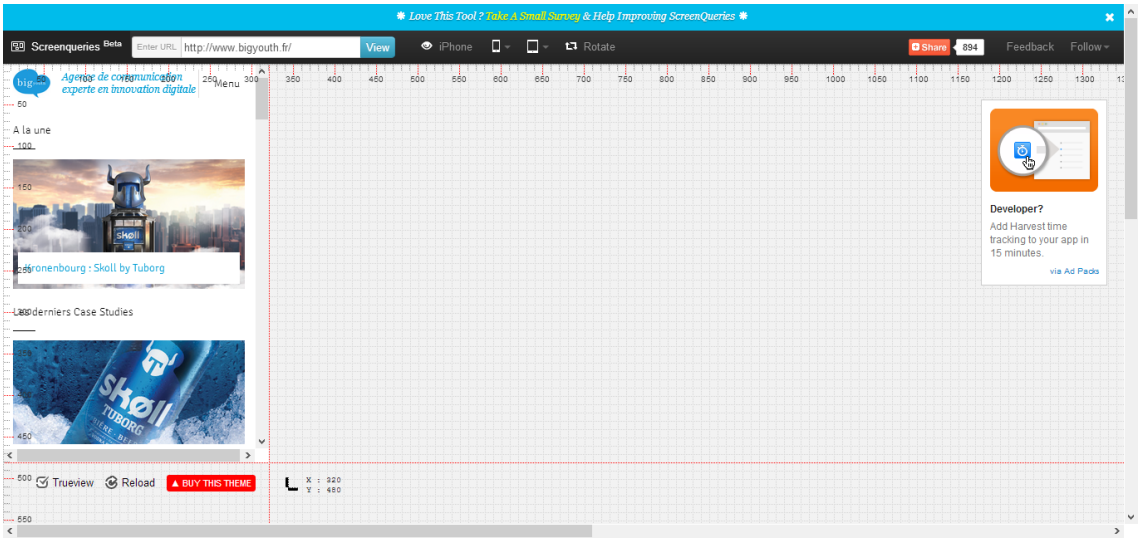


Ilustración 53Página web de Screen Queries

La funcionalidad de esta herramienta es similar a las anteriores, con la ventaja de poseer unas reglas que permiten redimensionar dinámicamente como deseemos. Quizás estas sean las más conocidas, la mayoría nos proporcionan la misma funcionalidad ya que persiguen el mismo objetivo y existen muchas herramientas más como Responsive Design Test (http://www.responsivedesigntest.net/), Responsive Is (http://responsive.is/), Responsive Px (http://responsivepx.com/) o http://mattkersley.com/responsive/.

4.9 Empleo de Frameworks para RWD

4.9.1 Concepto de un framework front-end

Un framework para front-end es un conjunto de conceptos y herramientas que nos facilita considerablemente el trabajo de diseñar una web proporcionándonos una base o esqueleto para nuestros nuevos diseños, como por ejemplos dándonos la posibilidad adaptarlo a distintas resoluciones y tamaños de pantalla a través de sistemas de grids u otros mecanismos.

La mayoría de los desarrollos web comparten una estructura simular, por lo tanto el objetivo de estos frameworks es proporcionar una estructura común para desarrolladores web, para agilizar el proceso de inicialización aportando reutilización de elementos básicos y repetibles.

Estos frameworks suelen consistir en una estructura de archivos y directorios de código estándar divididos en elementos html, css y javascript. En general la mayoría de estos frameworks comparten características como proporcionarnos un código css para diseñar nuestros layouts, lo que se conocen como grids o cuadriculas, suelen contener definiciones de tipografía, soluciones para el problema de las incompatibilidades de los distintos navegadores como reset css, y componentes avanzados de interfaces de usuario.

4.9.2 Ventajas y desventajas de emplear un framework front-end

El empleo de un framework posee una serie de ventajas para el desarrollador web, pero también inconvenientes, a continuación detallaremos las principales ventajas e inconvenientes de emplear este tipo de herramientas a la hora de diseñar nuestras páginas webs.

La principal ventaja del uso de estas herramientas es la rapidez que aportan debido a que no es necesario implementar todo de una manera manual y desde cero, este tipo de frameworks nos aporta ciertas funcionalidades ya implementadas proporcionando agilidad sobre todo a la hora de maquetar nuestros diseños. Además esta agilidad será bien recibida por nuestros clientes que siempre exigen el tiempo como mayor valor.

El gran apoyo de la comunidad web con ciertos de estos frameworks aporta una gran documentación y gran cantidad de desarrolladores compartiendo conocimiento y

buenas prácticas sobre estos, además estas herramientas aportan un código limpio y ordenado.

Aportan un nivel de abstracción superior a la hora de decidir factores como los saltos necesarios de mediaqueries para cada dispositivo e incluso algunas herramientas aportan soluciones a los problemas comunes en CSS y de compatibilidad de navegadores.

En contraposición a las ventajas de emplear un framework, describiré las desventajas de no emplear ningún framework y trabajar de manera manual.

Al trabajar con un framework se pierde en libertad, ya que debemos de ajustar nuestro diseño a un grid preestablecido y a los requerimientos de la herramienta. Sin embargo, creando nuestro propio diseño de forma manual creamos un código más flexible y siempre bajo control aunque haya que invertir mayor tiempo en su desarrollo. Además el empleo de un framework requiere de una curva de aprendizaje para manejarlo.

4.9.3 Framework a estudiar: Bootstrap

Bootstrap es algo más que un sistema de grids para desarrollar nuestras estructuras webs. Bootstrap es un conjunto de herramientas proporcionadas por los creadores de Twitter que nos aportan distintos widgets y estilos para desarrollar con gran agilidad el front-end de nuestras aplicaciones web. Las aplicaciones web actuales han alcanzado un nivel de abstracción tan alto como cualquier aplicación de escritorio teniendo en común ciertos elementos incluidos en cada proyecto, tanto en diseño como en funcionalidad.

Este framework nos abstrae de las compatibilidades entre navegadores poniendo a disposición del desarrollador un conjunto de elementos como pueden ser desde formularios, botones, tablas hasta menús, alertas y otros componentes que agilizan bastante nuestro trabajo.

Este kit de herramientas fue un proyecto interno de Twitter que posteriormente decidieron publicar totalmente abierto a toda la comunidad web y se define a sí mismo como un framework front-end limpio, intuitivo y con gran poder para aportar rapidez y facilidad al desarrollo web. Sin embargo, el punto más importante que destaca la mayoría de la comunidad web es lo realmente fácil que es aprender a usar este framework gracias a su buena documentación y a cantidad de ejemplos que aporta.

Ilustración 54 Página web Principal de Boostrap

En este documento presentaré ciertas de sus funcionalidades pero sobre todo me centraré en mostrar las funcionalidades que nos aporta a la hora de desarrollar layouts adaptables para implementar nuestra filosofía Responsive Design.

4.9.3.1 Introducción a Bootstrap

Bootstrap nos permite descargar su conjunto de herramientas customizadas dependiendo de nuestras necesidades. Concretamente nos permiten tres opciones:

- La primera opción nos permite descargar todo el paquete de herramientas completo, ya compilado y en sus versiones minimizadas, preparado para poder incluirse en tu proyecto.
- La segunda opción también nos permite descargar todo el paquete de herramientas completas, pero en este caso con el código fuente y la documentación completa para poder ser modificada, estudiada y optimizada por los desarrolladores.
- La última opción es descargarnos los componentes del framework que se vayan a emplear con el objetivo de personalizar y optimizar las necesidades requeridas por nuestra aplicación web.

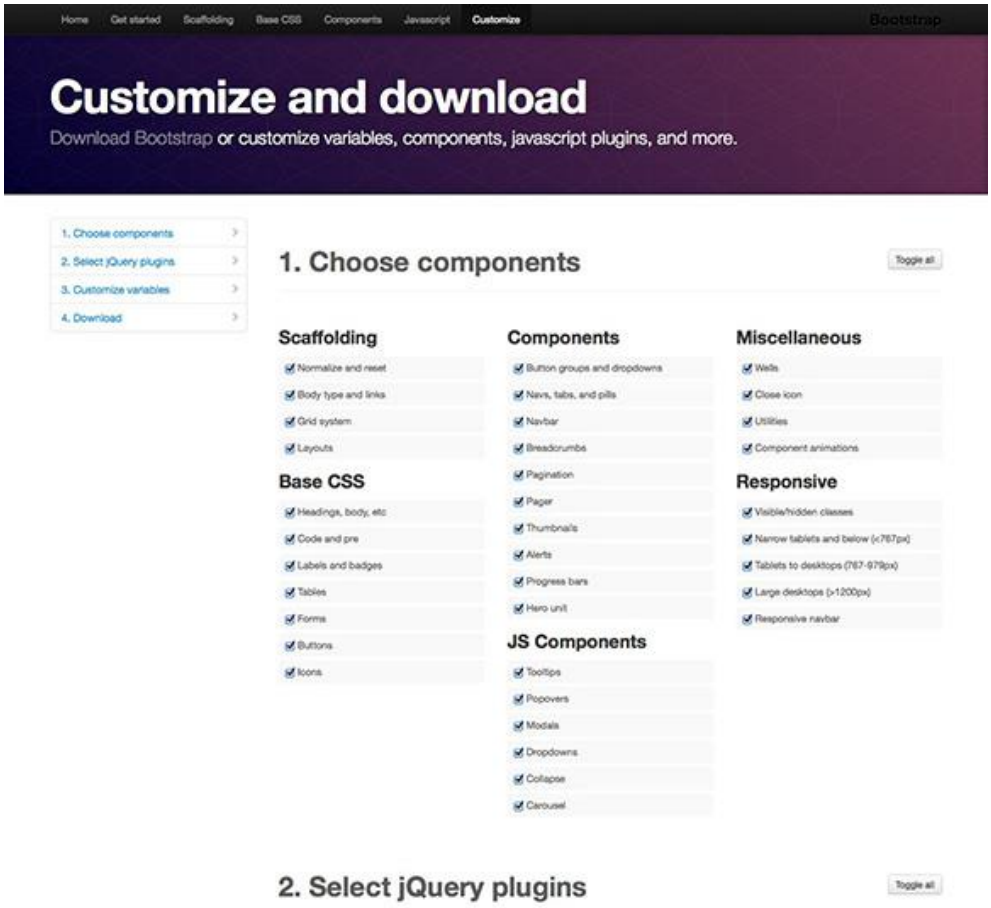


Ilustración 55 Ventana de customización de Boostrap

4.9.3.2 Estructura del framework

El código fuente descargado está estructurado en tres directorios con una pequeña cantidad de ficheros fácilmente reutilizables e integrables en nuestro proyecto. Concretamente nos encontraremos con los siguientes directorios:

- CSS: Esta carpeta contiene dos ficheros css más sus versiones minimizadas. Los ficheros son bootstrap.css y bottstrap-responsive.css. Estos ficheros se emplean para estimular los elementos de la web. La versión responsive incluye todos los componentes necesarios para incluirlos en el proyecto.
- JS: Esta carpeta incluye el fichero bootstrap.js además de su versión minimizada donde se encuentra todo el código javascript necesario para el correcto funcionamiento de los widgets de bootstrap.
- IMG: Esta carpeta incluye los sprites empleados para emplear los iconos de Bootstrap cedidos por Glyphicons.

```
- bootstrap/
  -- css/
    --- bootstrap.css
    --- bootstrap.min.css
    --- bootstrap-responsive.css
    --- bootstrap-responsive.min.css
  -- js/
    --- bootstrap.js
    --- bootstrap.min.js
  -- img/
    --- glyphicons-halflings.png
    --- glyphicons-halflings-white.png
```

4.9.3.3 Herramientas que aporta Bootstrap

Las herramientas disponibles descritas por la documentación oficial son:

- **Scaffolding**: En este apartado se describen las herramientas que Bootstrap nos proporciona para crear la estructura básica de la aplicación web. En este apartado se describe el sistema de grids tanto fijo como fluido. Posteriormente se estudiará cómo se comporta este sistema de grids.
- **Base CSS**: En este apartado se incluye una gran variedad de estilos CSS que se aplican por defecto a las etiquetas más básicas empleadas en HTML. Por ejemplo se describen todos los estilos que se emplearán a las cabeceras (h1,h2,h3,…), el estilo de los párrafos y tablas y de todos los elementos de un formulario. Como se ha comentado se descarga junto al framework un conjunto de sprites de Glyphicons. En este apartado se describe como emplear los 140 iconos. Para el correcto funcionamiento de estas herramientas el framework hace uso de la versión CSS3 para asegurar una buena experiencia de usuario y garantizando la mayor compatibilidad entre navegadores.
- **Componentes:** En este apartado se incluyen todas las herramientas de interfaz como menús desplegables, botones, barras que nos aporta este framework y fácilmente integrables en nuestra web con tan solo unas pocas líneas de código.
- **JavaScript:** En este apartado se incluyen ejemplos de código Javascript para cada uno de los componentes proporcionándonos el conjunto de propiedades y eventos que podemos emplear para el funcionamiento customizado de estos widgets. Se incluyen widgets como popovers, listas desplegables, pestañas, etc…

4.9.3.4 Incluir Bootstrap en nuestra aplicación web

Una vez conocemos la estructura y los componentes que nos aporta descubramos como podemos incluir el framework en nuestros proyectos web. Básicamente debemos de incluir los archivos descargados (CSS Y JS) y en cada página html incluyéndolos dentro de las etiquetas <head>. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
      <head>
        <meta charset="utf-8">
        <title>Bootstrap</title>
        <link rel="stylesheet" href="css/bootstrap.min.css">
<script
src="//ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
 <script src="js/bootstrap.min.js"></script>
 </head>

 <body>
    <p>!Hola Bootstrap!</p>

 </body>
</html>
```

4.9.3.5 Sistema de columnas de Bootstrap

Bootstrap proporciona un sistema de columnas, también denominada cuadrícula o rejilla, bastante fácil de usar a la hora de diseñar layouts y bastante sólido tanto para layouts fijos como fluidos.

El sistema se basa en un grid de 12 columnas y cubre un ancho de 940 pixels en su versión estática y entre 724 pixles y 1170 pixeles en su versión fluida. Bootstrap permite que nuestra aplicación responda adaptándose a cualquier dispositivo con un diseño fluido ocupando si es necesario el 100% de amplitud de pantalla o por otro lado que nuestra aplicación sea un diseño estático que no necesite el 100% del ancho del dispositivo.

Ilustración 56 Sistema de Columnas de Bootstrap

A continuación se detallan los distintos tamaños asociados a cada tipo de dispositivo con los que trabaja Bootstrap

Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and above	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

Ilustración 57 Breakpoints de Bootstrap

Bootstrap implementa su layout adaptable mediante el uso de mediaqueries. La documentación nos muestra los siguientes puntos de ruptura en nuestros diseños que se corresponden con la tabla anterior:

```
/* Large desktop */
@media (min-width: 1200px) { ... }
/* Portrait tablet to landscape and desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }
/* Landscape phone to portrait tablet */
@media (max-width: 767px) { ... }
/* Landscape phones and down */
@media (max-width: 480px) { ... }
```

Además de ajustar los tamaños a los dispositivos, Bootstrap nos proporciona para los tres niveles de dispositivos (mobile, Tablet y desketop) la posibilidad de mostrar u ocultar elementos de nuestro html dependiendo del tipo de vista. A continuación se muestra una tabla con las clases CSS necesarias:

Class	Phones 767px and below	Tablets 979px to 768px	Desktops Default
.visible-phone	Visible	Hidden	Hidden
.visible-tablet	Hidden	Visible	Hidden
.visible-desktop	Hidden	Hidden	Visible
.hidden-phone	Hidden	Visible	Visible
.hidden-tablet	Visible	Hidden	Visible
.hidden-desktop	Visible	Visible	Hidden

Ilustración 58 Mostrar y ocultar elementos en Bootstrap

Para que el sistema de grids funcione correctamente todo el contenido de la página debe incluirse dentro de un DIV aplicándole la clase CONTAINER que nos sirve un contenedor centrado en el navegador y estableciendo un ancho establecido para el contenido de la página. Dentro de este gran DIV contenedor se deben definir DIVS aplicándoles la clase ROW, que crean columnas dentro del wrapper manteniendo márgenes, padding y clear entre estas filas. Finalmente dentro de estos DIVS se implementa el diseño ocupando el espacio necesario a través de elementos que ocupan un espacio determinado aplicándoles la clase SPAN con determinado valor, el cual indica el número de columnas se emplea. Por ejemplo si deseamos 3 cajas en una fila debemos de dividir 12 columnas entre 3 cajas obteniendo columnas de tamaño span4.

```
<div class="container">
      <div class="row">
            <div class="span4"> 1  </div>
            <div class="span4">  2  </div>
            <div class="span4">  3  </div>
      </div>
</div>
```

A la hora de ocupar el espacio en columnas la suma de los elementos no debe exceder 12 columnas con el fin de evitar que se descuadre el sistema de Grids. Recursivamente dentro de cada elemento puede estar dispuesto de la misma manera con un máximo de tamaño de columnas del padre:

```
<div class="container">
      <div class="row">
```

```
            <div class="span4"> Sidebar </div>
            <div class="span8">
                  <div class="row">
                        <div class="span3"> 3 </div>
                        <div class="span2"> 2 </div>
                        <div class="span3"> 3 columnas </div>
                  </div>
            </div>
      </div>
  </div>
```

Estos ejemplos nos muestran cómo desarrollar layouts fijos estáticos, pero a continuación se mostrarán las características que nos aporta Bootstrap para desarrollar el diseño de nuestros layouts de una forma adaptable sin que los tamaños de columna sean fijos. Para emplear este layout fluido se debe incorporar a la página el fichero bootstrap-responsive.css. Además la estructura sería idéntica que en los casos anteriores solo se debe sustituir la clase CONTAINER por CONTAINER-FLUID y la clase ROW por ROW-FLUID. De esta manera se distribuirá idénticamente dentro de cada capa. Sin embargo, en este caso los anchos se ajustarán al ancho establecido por el dispositivo.

```
<div class="container-fluid">
      <div class="row-fluid">
            <div class="span4"> 1 </div>
            <div class="span4"> 2 </div>
            <div class="span4"> 3 </div>
      </div>
</div>
```

Si en el grid fijo los elementos interiores debían sumar hasta el número de columnas definidas por el padre, en este caso los elementos internos deben detener como resultado el valor 12, así si deseamos que dentro de una columna un elemento ocupe la mitad del espacio se deberían de aplicar la clase span6.

```
<div class="container-fluid">
      <div class="row-fluid">
            <div class="span4"> 4/12 </div>
            <div class="span8">
                  <div class="row-fluid">
                        <div class="span5"> 5/12 </div>
                        <div class="span2"> 2/12 </div>
                        <div class="span5"> 5/12 </div>
                  </div>
            </div>
      </div>
</div>
```

4.9.3.6 Ejemplo de un Diseño Responsive empleando Bootstrap

A continuación mostraré un ejemplo sencillo de un diseño Responsive Web Design con Bootstrap explicando los componentes más importantes de éste. Primero vamos a definir las estructuras HTML a utilizar empleando las técnicas de Bootstrap para crear un layout responsive.

Primeramente añadiremos las hojas de estilo y los javascript necesarios para implementar las funcionalidades Responsive. Además en la etiqueta meta viewport, definiremos el ancho que tome el navegador con el ancho del dispositivo.

```
<head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
      <title>Bootstrap Responsive</title>
      <link rel="stylesheet" href="css/style.css">
      <link rel="stylesheet" href="css/bootstrap.css">
      <link rel="stylesheet" href="css/bootstrap-responsive.css">
      <script src="http://code.jquery.com/jquery-1.9.0.min.js"></script>
      <script src="js/bootstrap.js"></script>
</head>
```

A continuación se desea insertar una barra de navegación en la parte superior. Para implementar una barra de navegación Responsive primeramente debemos de definir un div aplicándole la clase navbar y navbar-fixed-top, para que siempre se quede fija en la parte superior de la web. Seguidamente debemos de crear una capa donde se encontrará todo el contenido interno de esta barra de navegación con la clase navbar-inner. Como queremos que el contenido se encuentre centrado con el wrapper encapsulamos los elementos en una capa con la clase container.

Este es el momento de determinar los elementos de la barra de navegación. Primeramente definiremos un brand con el nombre de la página web. Seguidamente definimos una serie de elementos de navegación. Bootstrap por defecto nos aporta la funcionalidad de transformar la lista de elementos nav en un botón denominado collapse el cual encapsula el menú en un botón desplegable empleando la técnica de Off Canvas cuando las dimensiones son de tablet o smartphone. Con este código y definiendo en los atributos data-toggle y data-target la capa donde se encuentra la navegación obtendremos una barra de navegación totalmente Responsive.

```
<div class="navbar navbar-fixed-top">
      <div class="navbar-inner">
            <div class="container">
                  <a href="#" class="brand">responsive pfg</a>
                  <a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
                        <span class="icon-th-list"></span>
                  </a>
                  <div class="nav-collapse collapse">
                  <ul class="nav" >
                        <li class="active"><a href="#">Inicio</a></li>
                        <li><a href="#">Experiencia Laboral</a></li>
                        <li><a href="#">Formación Académica</a></li>
                  </ul>
                  </div>
            </div>
      </div>
</div>
```

El siguiente elemento a insertar es un componente que nos proporciona Bootstrap denominado Hero Unit, con tan solo aplicarle la clase hero-unit a un div obtendremos este componente. Además le incorporamos al clase hidden-phone para que no se vea en la versión móvil.

```
<div class="hero-unit fondo-hero hidden-phone">
      <div class="container">
            <h1 class="titulo center">adrianAlonsoVega()
                  <h2 class="center subtitulo">
                        { ingeniero informático }
                  </h2>
            </h1>
      </div>
</div>
```

Vayamos a la parte importante del diseño, el layout que marcará la distribución de los elementos. Como hemos comentado dentro del contenedor debemos definir las columnas fluidas que tendrá cada elemento usando la clase row-fluid. El elemento de la primera capa solo se verá en smartphones gracias a la clase visible-phone. Luego se ha definido otra fila la cual se reparte del espacio disponible de 12 columnas en una columna en la izquierda de 4 sobre 12 y la parte de la derecha de 8 sobre 12.

En la columna de la izquierda se inserta una imagen y un texto dentro de las etiquetas article. Al encontrarse dentro de un row-fluid esta imagen escalará al reducirse o aumentar el ancho de la columna.

La columna de la derecha posee interiormente otras dos filas. La primera fila contendrá dos columnas que se repartirán el espacio disponible al 50%, por ello se le asigna una clase span6 a cada una, para repartirse las 12 columnas relativas de su contexto. Cada una de estas columnas contendrán una imagen, un encabezado y un texto.

La segunda fila ocupará se dividirá igualmente, salvo que en la columna de la izquierda solo poseera una imagen que escalará proporcionalmente y en la parte derecha un texto.

```
<!-- Layout -->
<div class="container">
            <div class="row-fluid">
                  <h1 class="visible-phone">Adrian Alonso Vega</h1>
            </div>
            <div class="row-fluid">
                  <div class="span4">
                        <img src="img/silueta.gif" class="span12 center
silueta hidden-phone" />
                        <article class="descripcion"> Texto
                        </article>
                  </div>
                  <div class="span8">
                        <div class="row-fluid">
                        <article class=" span6">
                        <img src="img/bs-docs-twitter-github.png"
class="marketing-img">
                              <h2>
                                    ¿Quien Soy?
                              </h2>
                                    <p> texto </p>
                        </article>

                        <article class=" span6">
                        <img src="img/bs-docs-responsive-
illustrations.png" class="marketing-img">
                              <h2>
                                    ¿Que ofrezco?
                              </h2>
                                    <p>texto</p>
                        </article>
                  </div>
            <div class="row-fluid">
                  <article class="span6">
                        <img src="img/bs-docs-bootstrap-features.png"
class="marketing-img">
                  </article>
                  <article class=" span6">
                        <h2>
                              ¿Qué ofrezco?
                        </h2>
                        <p>texto</p>
                  </article>
            </div>
      </div>
</div>
```

Con esta simple estructura HTML hemos marcado como se desea que sea nuestro layout básico y solo faltarían pequeños detalles CSS ya que el propio framework será el encargado de reorganizar nuestros elementos. Sin embargo, para los saltos definidos podemos manipular los tamaños de fuente para decidir cómo queremos que se aprecien.

```
@media (max-width: 980px) {
```

```
        .titulo{
                font-size: 2em !important;
        }

        .subtitulo{
                font-size: 1em !important;
        }
}

@media (max-width: 480px) {
        .titulo{
                font-size: 1.6em !important;
        }

        .subtitulo{
                font-size: 1em !important;
        }

        .marketing-img {
            min-width: 300px;
        }
}
```

Ilustración 59 Version Desktop del Ejemplo con Bootstrap

Comprobemos el resultado de nuestra página web ejecutándose en un navegador de una Tablet:

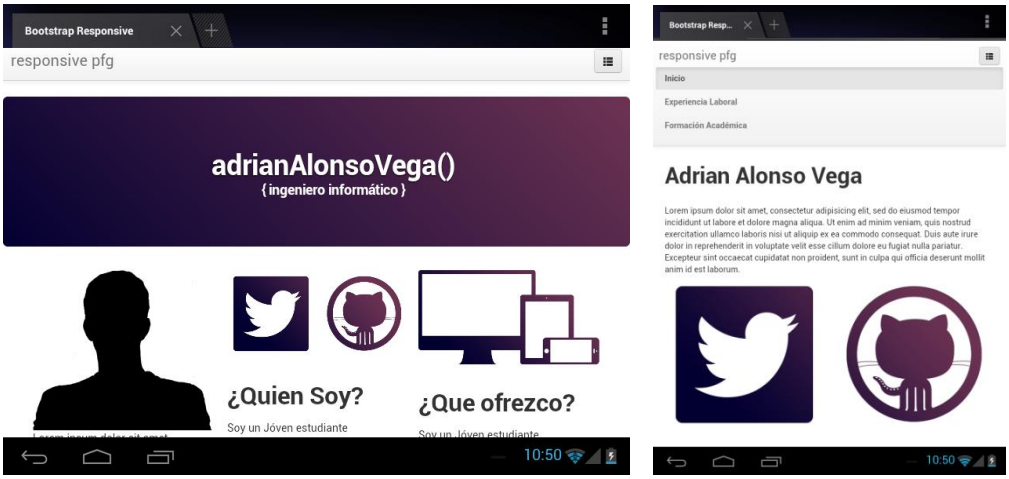


Ilustración 60 Versión tablet del ejemplo en Bootstrap

Además comprobamos como se ve desde un smartphone:

Ilustración 61 Versión Mobile del ejemplo de Boostrap

4.9.3.7 El futuro de Bootstrap

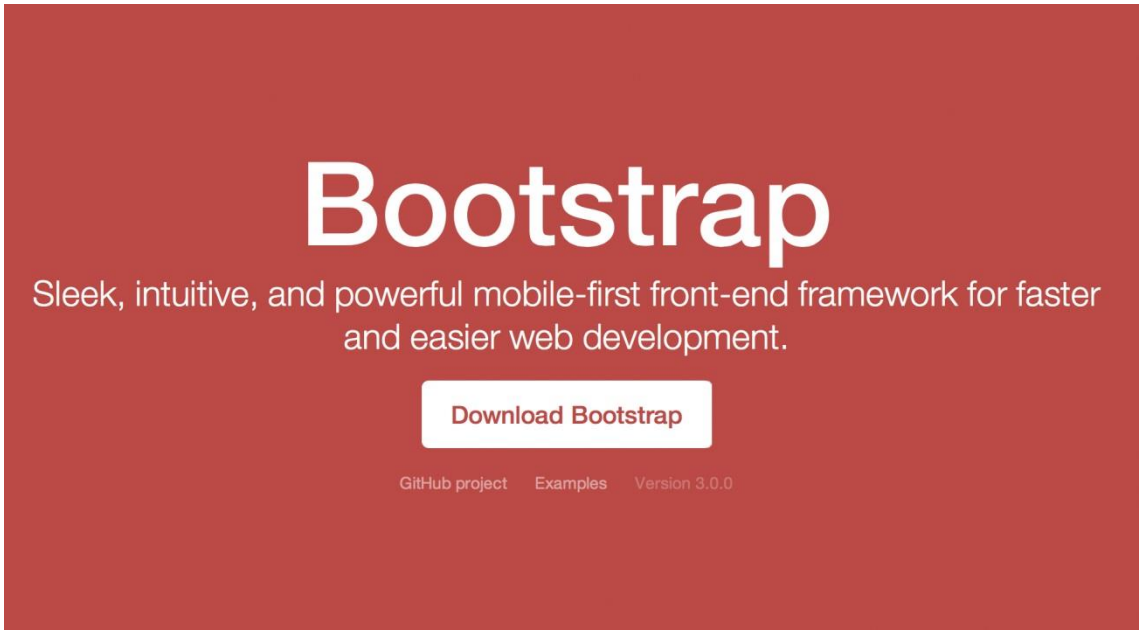


Ilustración 62 Futura Página Principal de Bootstrap 3

Todo lo descrito en el apartado anterior es lo que actualmente nos proporciona la versión 2.3.2 de Bootstrap pero actualmente se encuentra en desarrollo la nueva versión del framework 3.0. Se han confirmado un gran número de funcionalidades nuevas y mejoras de las ya existentes que nos marcarán la forma de desarrollo del front-ent en un futuro más bien próximo, que se alinea totalmente con la forma de pensar de Responsive Web Design.

Las últimas actualizaciones han sido una reestructuración de la documentación, CSS y javascript. Sin embargo, la versión 3.0 es un proyecto más ambicioso y los creadores han confirmado que perseguirá el concepto de moble-first. Además han confirmado que se expondrán ejemplos para verse reflejado como se debe implementar este punto de vista, por lo que por fin la funcionalidad Responsive se incluirá en el núcleo del framework de tal manera que ya no se encontrará en ficheros externos.

Respecto a la compatibilidad con los navegadores se dejará de dar soporte a Internet Explorer 7 y a Firefox 3.6 eliminando todos los hacks que permitían la compatibilidad de estos navegadores.

El cambio más importante que se ha anunciado es el cambio drástico del sistema de grid que será más simple y potente, además en esta ocasión se ha unificado en un único grid la funcionalidad fluida que se encuentra separada en la versión actual. Por defecto vendrá establecido el box-sizing al valor border-box para acomodar los cálculos de anchos al diseñador. Este grid perseguirá la filosofía mobile first, por lo que empezará apilando los div y escalando según aumente el tamaño a través de mediaqueries, una contraposición al sistema de grid actual que parte de un layout de 960 px y escala hacia

abajo. La semántica de clases CSS también cambiará pues sustituirán los famosos spanx por col-span-x.

Además de todo esto se proporcionarán parámetros para que determinemos los breakpoints que a nosotros más nos interesen. Más características interesantes que nos facilitarán el Responsive es que todos los elementos de formulario que se nos ofrecen se adaptarán por defecto al contenedor padre.

Sus creadores buscan ser amigables con las versiones mobile desde el principio, sobre todo por la competencia ya que otros populares frameworks web ya han evolucionado y la mayoría quiere dar soporte al crecimiento de los dispositivos móviles, ya que es muy importante para el futuro de la web.

4.9.4 Otros Frameworks Front-end

Además de Boostrap existen multitud de frameworks front-end y entre los más conocidos que nos proporcionan herramientas para aplicar Responsive Web Design se encuentran:

Skeleton

Skeleton es una colección de ficheros CSS que ayudan a desarrollar sitios rápidamente en cualquier tamaño de resolución. Este framework posee un sistema de grids de 960 pixeles, que escala hacia abajo para adaptarse a cualquier tipo de dispositivo. Además se define a sí mismo como una herramienta de desarrollo ágil persiguiendo buenas prácticas CSS pero no se considera un framework de interfaz, solo provee las estructuras básicas para comenzar con bastante agilidad.

Foundation 4

Este framework se define a sí mismo como el framework front-end Responsive más avanzado del mundo y es fiel competidor del framework Bootstrap. Incluye un sistema de grids de 12 columnas fluidas similar a Boostrap con un conjunto de clases CSS mucho más semánticas y entendibles, además propone gran cantidad de ejemplos de layouts para realizar un rápido prototipado para los clientes. Igualmente contiene componentes similares a Bootstrap para iconos, botones, formularios, elementos de navegación o tipografías. Una buena opción pero con una curva de aprendizaje inicial que requiere un esfuerzo para el desarollador.

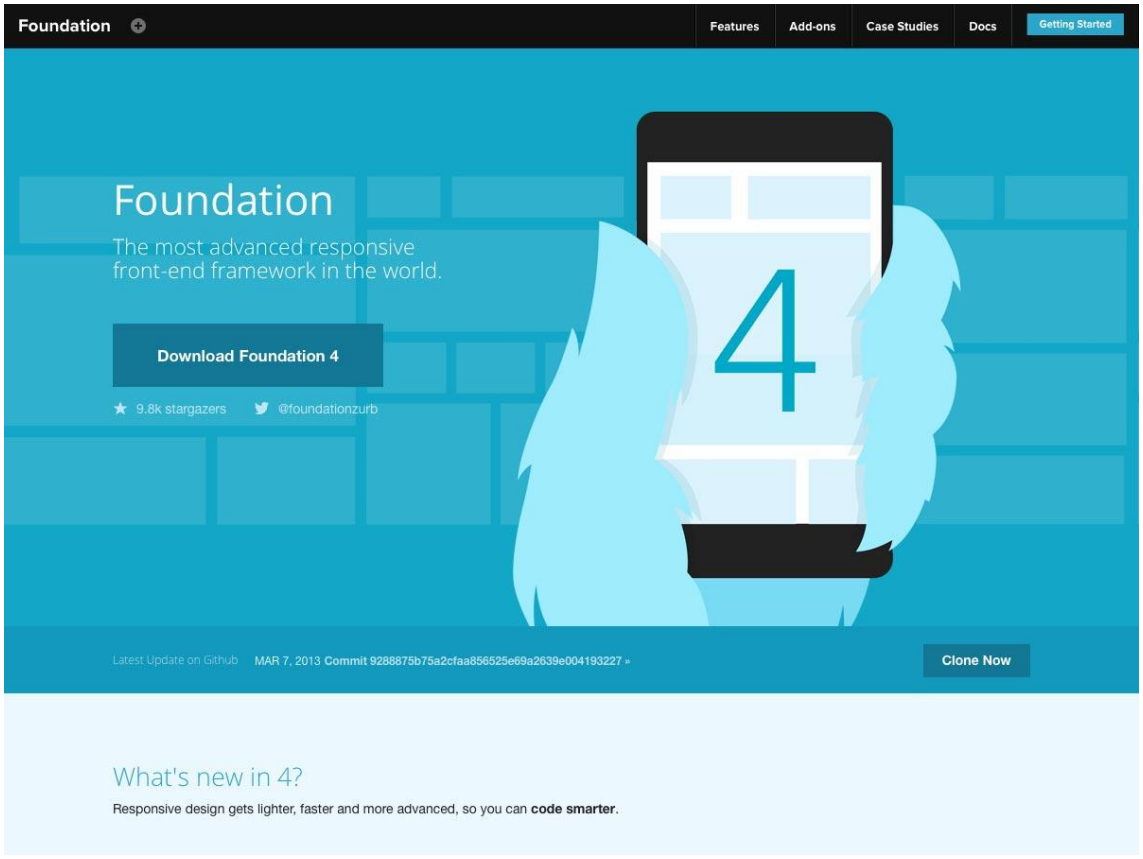


Ilustración 63 Página web de Foundation

4.9.5 Conclusiones

El empleo de Boostrap framework aporta facilidad de uso, agilidad de desarrollo, reutilización del código aportando un sencillo mantenimiento y aplicaciones ejecutables en cualquier navegador.

Algunos desarrolladores apoyan totalmente el empleo de frameworks. Sin embargo, otros prefieren trabajarse su diseño manualmente desde cero. La decisión del empleo de un framework no necesariamente debe ser de un desarrollador, este arbitraje supone un equilibrio entre las necesidades del proyecto, la experiencia del desarrollador y de la finalidad del proyecto.