Responsive web design

PID_00254184

Marcos González Sancho

Universitat Oberta de Catalunya

Índice

1. Introducción

Este documento recoge los fundamentos del responsive web design, como técnica de desarrollo web, para crear sitios que se adaptan a los diferentes contextos de los dispositivos desde los que se visualizan.

El documento no se centrará específicamente en ningún framework frontend particular que base su funcionamiento en dicha técnica, sino que afrontará el análisis de esta desde una forma aislada y básica, para poder centrarse en los pilares que la sustentan, y evitar así una complejidad innecesaria.

2. Historia y definición

El concepto de responsive web design (en adelante, RWD) fue introducido en la escena del desarrollo web por Ethan Marcotte en mayo del 2010, en un artículo de la conocida *A List Apart*, que ya en el 2009 había publicado otro artículo muy relacionado relativo al uso de rejillas fluidas. Desde la introducción del concepto ha sido uno de los aspectos del desarrollo frontend sobre los que más se ha escrito y comentado, como demuestra el hecho de que desde el 2012 ha estado siempre en los primeros puestos de los ránquines de tendencias en diseño web, como en el ranquin de Forbes para el 2014, en el que aparece en primer lugar.

Existen varias definiciones para RWD, algunas muy centradas en el tamaño de visualización del dispositivo y otras que abarcan conceptos mucho más generales y complejos. Recogemos a continuación algunas que nos servirán de referencia para el contenido que desarrollaremos en las siguientes páginas.

2.1. Definición por Ethan Marcotte, en el 2010

La definición que se puede extraer del artículo en el que Ethan Marcotte acuña el término es la siguiente:

> En lugar de ajustar diseños no relacionados para cada uno de la multitud de dispositivos existentes que acceden a la web, podemos tratar esos diseños como facetas de una misma experiencia. Podemos diseñar para una experiencia de visualización óptima, pero incluyendo tecnologías estándar en nuestros diseños para hacerlos no solamente más flexibles, sino más adaptados al medio que los muestra.

2.2. Definición por Kayla Knight en *Smashing Magazine*

La definición que esta autora refleja en la prestigiosa publicación *Smashing Magazine* amplía el abanico de aspectos a contemplar con esta técnica:

> RWD es la técnica que sugiere que el diseño y el desarrollo deben responder al comportamiento y entorno del usuario, basado en el tamaño de pantalla, plataforma y orientación.

2.3. Definición por Wikipedia

La traducción al castellano del término *responsive* no está demasiado clara, entrando en escena términos como *responsivo* (perteneciente o relativo a la respuesta según la RAE), *adaptable*, *adaptativo*... en cualquier caso en la Wikipedia se recoge como *diseño web adaptable*, y se define como:

> El diseño web adaptable o adaptativo (en inglés, Responsive Web Design) es una filosofía de diseño y desarrollo web que mediante el uso de estructuras e imágenes fluidas, así como de media-queries en la hoja de estilo CSS, consigue adaptar el sitio web al entorno del usuario.

Es curioso que no se suele utilizar el término *sensible* cuando es una de las traducciones literales que se podrían aplicar del término inglés, y que sin duda tiene mucho de apropiado cuando se abrazan las definiciones más amplias que existen detrás del RWD.

3. Responsive web design (RWD)

Todas las definiciones anteriores sumadas a la evolución del propio término durante estos últimos años, y entendidas desde la visión de que el RWD va mucho más allá de la gestión de contenido frente a los diferentes escenarios de tamaños de pantalla, nos llevan a una definición más completa del término:

> Se puede entender RWD como la combinación de enfoque, estrategia y técnicas que tienen como objetivo crear una experiencia unificada y consistente para los usuarios que acceden a los contenidos, independientemente del dispositivo con el que acceden, y más allá aún del dispositivo, del escenario o contexto en el que acceden.

Esta definición abarca conceptos que se tratarán más adelante, tales como progressive enhancement, mobile first, rendimiento, optimización, etc.

El contexto de un usuario va más allá de las dimensiones del área visible de su dispositivo o de un punto de corte aplicado mediante media queries en nuestro planteamiento responsive del sitio web. El contexto de un usuario referencia factores como la forma de interacción, capacidades del dispositivo, la conectividad, condiciones del entorno (como el tiempo disponible, el lugar), aspectos culturales del usuario, etc. Esta interpretación de contexto es muy similar a la aplicada en las denominadas aplicaciones contextuales (1 y 2), que dotan de una experiencia continua, unificada y adaptada al usuario durante el uso de una aplicación desde diferentes contextos.

Como es lógico, querer enfrentarse a la totalidad de los aspectos que definen un contexto es realmente complejo, ya que además de ser complicado acotar o limitar todos los factores que influyen en el usuario en ese contexto, algunos de esos factores tienen difícil solución hoy en día por las propias limitaciones de la tecnología.

4. Causas de su creciente adopción

Las principales causas por las que el RWD ha sido adoptado de forma notable por la comunidad de diseñadores y desarrolladores abarcan diferentes aspectos, en la misma línea de la definición que hemos establecido. Existen causas de enfoque o filosofía sobre cómo debe ser una web universal y accesible desde cualquier dispositivo, causas estratégicas a nivel de desarrollo y productividad y causas técnicas. Veremos a continuación algunas de ellas.

4.1. Aumento de la demanda

El consumo de contenidos en internet a través de dispositivos está sufriendo un crecimiento espectacular en los últimos años, y esto ha llevado al desarrollo web a una nueva realidad en la que la compatibilidad con dispositivos es una obligación.

Ante esta necesidad, ya en el 2005 se estudiaban las diferentes aproximaciones que se podían hacer, entre las que aparecía la posibilidad de emplear `media=«handheld»` para servir una hoja de estilos diferentes, como un mecanismo precursor de las actuales media queries.

En la actualidad, el mercado ya comienza a ser consciente de la importancia de ofrecer una experiencia satisfactoria para los usuarios que acceden desde estos dispositivos (ellos mismos la experimentan) y, por tanto, los clientes lo demandan.

4.2. Un contenido único para todas las versiones

El contenido es el foco principal del desarrollo y es único, por lo que, al disponer de este unificado para todas las versiones de la web, se reduce parte del proceso de desarrollo. Esta ventaja tiene diferentes implicaciones que derivan de ella:

1) Contenido con una URL única siempre.

2) Contenido compartido siempre accesible de forma correcta, independientemente del dispositivo desde el que se compartió y desde el que se accede.

3) Mayor facilidad en la gestión de las analíticas web.

4) Menor coste en el mantenimiento del sitio.

4.3. Mejor rendimiento SEO

Desde mediados del 2012, Google recomienda de forma explícita esta aproximación en su «Cómo crear sitios webs optimizados para móviles», como consecuencia lógica del anterior punto, ya que el buscador premia el contenido frente a cualquier otro aspecto.

4.4. Ahorro de tiempo y costes al no tener que desarrollar sitios separados

Aunque este punto es bastante discutible frente a aproximaciones como el desarrollo específico para dispositivos, generalmente se considera menos costoso crear diferentes versiones adaptadas de un mismo contenido, que versiones separadas y relativamente independientes para diferentes contextos.

4.5. No dependencia de técnicas con detección de dispositivo

Al contrario que otras estrategias para crear sitios web para dispositivos, el RWD no hace uso de la detección de dispositivo (normalmente a través del user-agent), que es un proceso que resulta bastante problemático, ya que los user-agents son un aspecto complejo de gestionar correctamente por su propia complejidad y la continua aparición de nuevos dispositivos y nuevos user-agents (1 y 2).

5. Principales retos

El RWD no es una solución mágica a todos los problemas subyacentes ante el consumo de contenidos, desde la variedad de dispositivos y contextos. Como todas las soluciones, no deja de ser una opción ante un compromiso entre resultados, recursos y tiempo.

Cada proyecto es diferente y requiere un estudio para determinar qué aproximación es la más adecuada para él, y aunque, como acabamos de ver, el RWD permite resolver algunos aspectos clave, tiene aún importantes problemas que no han sido resueltos de forma completamente satisfactoria. Citamos algunos de los más importantes:

5.1. Rendimiento y optimización

Muchos de los problemas más importantes del RWD se encuentran en el hecho de que técnicamente no se han solucionado aspectos como la carga de contenido, que en algunos contextos no será visible, las peticiones al servidor que generan dicho contenido no visible, peso de las imágenes cargadas (al aplicar sobre ellas un escalado entre diferentes puntos de corte), funcionalidades que perjudican el rendimiento en dispositivos menos potentes al estar presentes en versiones para dispositivos superiores, etc.

Para solventar muchos de estos aspectos, el RWD abraza, como ya se ha comentado, la filosofía mobile first y el progressive enhancement, que básicamente lo que implica es comenzar trabajando para los dispositivos más limitados (generalmente móviles), e ir aplicando capas de mejoras de forma progresiva para dotar de una mejor experiencia a los dispositivos más avanzados, de manera que los requerimientos básicos y el rendimiento mínimo vayan de la mano de los dispositivos más limitados.

Estas consideraciones incluyen la optimización de los recursos cargados, tales como la reducción del número de peticiones combinando ficheros (CSS, JavaScript, spritesheets para imágenes…), minimización de ficheros (CSS y JavaScript), optimización de imágenes (compresión, uso de vectores, icon fonts, etc.).

5.2. Dificultad técnica de su implementación

Por los aspectos anteriormente expuestos, la implementación correcta de un sitio web bajo RWD no es una tarea especialmente sencilla. A nivel técnico implica un soporte más avanzado frente a diferentes navegadores de un mis-

mo contenido, y requiere una adaptación inicial de la forma de pensar en el desarrollo de los sitios web a través de la citada filosofía mobile first y del progressive enhancement.

Este cambio de mentalidad, arrancando desde el contenido puro, es algo que requiere entrenamiento y práctica, y que a través de la experiencia real en el desarrollo, permitirá ir asimilándolo cada vez en una fase más temprana del desarrollo.

5.3. Experiencia de usuario

Existen determinados elementos de experiencia de usuario y de interacción que requieren cambios importantes a la hora de mostrarse en los diferentes contextos (por ejemplo, la navegación) y que no son de resolución técnica directa o sencilla mediante solamente CSS o CSS + JavaScript si se quieren obtener buenos resultados.

5.4. Tipos de contenidos especialmente problemáticos

Algunas tipologías de contenidos web como imágenes, tipografía, tablas, canvas, mapas y cualquier otro contenido embebido, formularios, publicidad (banners)... son especialmente problemáticos para ser gestionados de forma óptima al aplicar RWD, y si bien existen avances y soluciones para algunos de ellos, en ocasiones no son óptimas.

6. Conceptos relacionados

Como hemos podido ver, existen multitud de conceptos que están íntimamente ligados a la aplicación del RWD hoy en día, o bien porque actúan de forma complementaria para lograr el objetivo principal del RWD, o bien porque son muy cercanos a nivel técnico.

6.1. Diseño estático, líquido, adaptativo y responsive

A nivel visual, el desarrollo web ha estado muchos años anclado en el diseño estático (con un ancho fijo para los contenidos, independientemente del dispositivo desde el que se visualizaban), aunque desde el inicio se contaba con la posibilidad de realizar un diseño líquido o fluido que basaba todos los anchos de sus contenidos en porcentajes, de manera que los contenedores y estructuras de una web se ajustaban al ancho disponible.

Con la llegada de las media queries comenzó a aplicarse el diseño adaptativo y responsive, sobre el que algunos autores hacen una diferenciación en el sentido de que el primero es una combinación de media queries con diseño estático (en ese rango que establecen las media queries) y el segundo combina la adaptación del layout a través de media queries con el diseño fluido.

6.2. Progressive enhancement

Tras una época en la que se daba por adecuada la técnica denominada «Graceful degradation», por la que algunos elementos web perdían capacidades de forma escalonada sin dejar de lado su funcionalidad, en el 2003 aparece de la mano de Steve Champeon y Nick Finck un concepto con una perspectiva bastante opuesta, al que se etiquetó como progressive enhancement (1 y 2).

Esta estrategia de desarrollo promueve centrarse inicialmente en el contenido, para, posteriormente, ir aplicando capas adicionales de presentación (mediante CSS) y funcionalidad e interacción (mediante JavaScript).

6.3. Mobile first

Esta filosofía acuñada bajo el término *mobile first* (1 y 2) por Luke Wroblewski defiende el hecho de enfocar el desarrollo atacando primeramente el contexto móvil, para dar soporte mejorado a la inminente oleada de accesos desde dispositivos móviles, para ayudarnos a aplicar los esfuerzos de base sobre el contenido e interacciones realmente importantes y para adaptar nuestros desarrollos a las nuevas capacidades que incorporan dichos dispositivos.

6.4. RESS

RESS es una técnica que ha aparecido posteriormente al RWD, es una combinación de este con componentes del lado de servidor. Su nombre viene de REsponsive Web Design + Server Side Components. Este enfoque o aproximación al diseño web trata de solventar algunos de los problemas más habituales del RWD desde el lado de servidor, para lograr mejorar la base del RWD.

Basa su funcionamiento en disponer de una plantilla con componentes, que en función de una detección de dispositivo mediante user-agent (con sus problemas), aplican una salida optimizada para dicho dispositivo, siendo cada componente responsable de la gestión de su «formato y comportamiento» en cada escenario.

7. Principales elementos del RWD

Como hemos ido definiendo, RWD hoy en día tiene muchas implicaciones y connotaciones que abarcan campos muy dispares, desde aspectos puramente técnicos y referentes al uso de lenguajes, características del navegador, etc., a la filosofía a la hora de abordar un proyecto web, optimización, etc.

En este apartado nos vamos a centrar en las partes fundamentales:

- Meta viewport
- Media queries
- Modelo de caja: border-box;
- Sistemas de rejilla
- Texto / Tipografía
- Imágenes

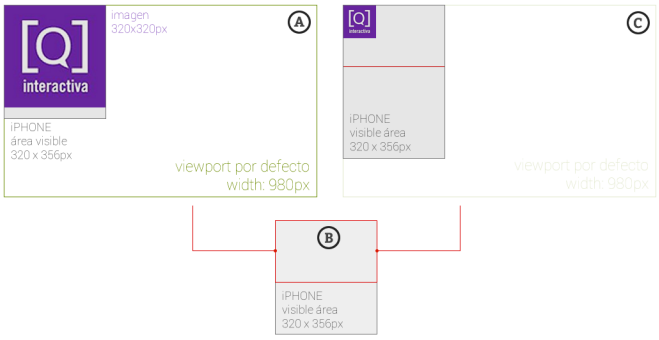
7.1. Meta viewport

La etiqueta meta «viewport» es un elemento fundamental para el correcto renderizado de una web realizada con RWD, o específicamente creada para dispositivos, en dispositivos móviles reales. Este viewport representa el área física que el navegador interpretará como necesaria para visualizar el contenido que está mostrando.

El viewport en el escritorio y en los dispositivos se comporta de manera notablemente diferente. En el escritorio es el área visible de la página, por lo que, cambiando el tamaño de la ventana, estamos cambiando el tamaño del viewport. En cambio, en los dispositivos el viewport puede ser más grande o más pequeño que el área visible, ya no coinciden.

Cada navegador móvil puede tener un valor por defecto diferente para el viewport, lo que implica que si queremos que estos navegadores interpreten correctamente cómo funciona nuestro contenido en diferentes anchos, tenemos que especificar esta etiqueta. De otra forma, el navegador móvil aplicará lo que él supone como viewport por defecto, y el resultado más habitual será encontrarnos con que el contenido se muestra completo en la pantalla del dispositivo, pero a consecuencia de aplicar una reducción de zoom importante.

A continuación se muestra un esquema de esta situación que acabamos de describir:



Paso A - B

El contenido se escala de los «interpretados» 980 px a los 320 px (aprox. escalándolo al 32 %) para que pueda mostrarse completo en el ancho del espacio visible del dispositivo.

Paso B - C

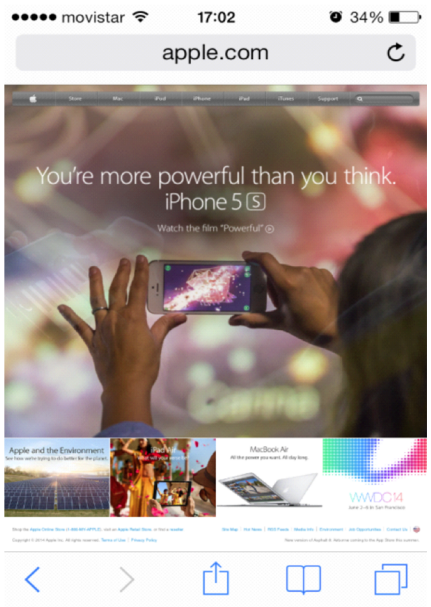
Esto provoca la clásica vista en miniatura que requiere interacciones de zoom in y zoom out, y que se suele producir en sitios web que no han definido el viewport de forma explícita o lo han definido a un tamaño de escritorio. En el esquema se puede apreciar que la imagen, a pesar de medir originalmente lo mismo que el área visible, se muestra mucho más reducida (al 32 %) para lograr mostrar todo un documento de unos supuestos 980 px de ancho.

Apple fue el creador de esta etiqueta (que no forma parte del estándar y que han adoptado muchos otros navegadores), que introdujo en la versión para móviles de Safari, para responder a la realidad de los navegadores móviles que muestran las páginas en una «ventana virtual» denominada viewport, y que generalmente es más ancha que la pantalla del dispositivo (actualmente existen multitud de tamaños de viewport). De esta manera, se muestra el contenido escalando el conjunto del viewport hasta que encaje con el tamaño físico real de la pantalla del dispositivo.

Este comportamiento es el que hace que una web convencional (sin especificar un meta viewport, o especificando uno con una medida fija), se muestre de una forma similar a la de los siguientes ejemplos:

Ausencia de *viewport*

Ausencia de *viewport*



Viewport fijado a 1024 px

Revista *Mosaic* con meta *viewport*

Revista *Mosaic* sin meta *viewport*

Esta mecánica es la que posteriormente permite al usuario manejar los sitios web creados para pantallas de escritorio en dispositivos, mediante zoom y desplazamiento en las diferentes áreas de la página.

Con el meta viewport los desarrolladores tienen la posibilidad de indicar al navegador el tamaño de esa «ventana virtual» y, por tanto, gestionar correctamente este comportamiento, tanto en tamaño como en escalado. Una lectura muy recomendable sobre el comportamiento y bases del viewport es la que se

puede encontrar en la web quircksmode.org (1 y 2), que complementa bien con algunos casos de test que ayudan a ver el comportamiento en un móvil real.

El uso habitual y recomendado para compatibilizarlo correctamente con RWD es el siguiente:

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

Como podemos ver, los diferentes atributos van separados por comas, y es importante que sea este carácter, ya que algunos navegadores no interpretan bien otros. Algunos desarrolladores añaden un «maximum-scale=1», pero al igual que «user-scalable=no» no es una decisión recomendable, ya que implica que el usuario pierde la posibilidad de modificar la escala de la vista y por tanto perdemos en accesibilidad. Existe un bug reconocido en iOS frente a los cambios de orientación con esta configuración, para el que también existe una solución o fix.

Finalmente, es importante tener en cuenta que IE10 ha elegido otro camino para dar solución a este comportamiento mediante el denominado CSS Device Adaption, que va acorde con lo propuesto por el W3C.

7.2. Media queries

Las media queries (incluidas en la especificación CSS3) son probablemente el componente más comúnmente asociado al RWD (y con un buen soporte en navegadores además de *polyfills* para IE6-8), y nos permiten aplicar diferentes reglas en las hojas de estilo en función de unas condiciones que se definen a través de ellas tales como anchos, altos, orientaciones, aspect ratios, etc. Por lo tanto, mediante estas podemos ajustar el layout y elementos visibles en nuestro contenido en función de dichos factores.

Las media queries se pueden aplicar (hablando de su sintaxis) a la hora de definir la etiqueta HTML de una hoja de estilos (a), como parte de una regla import en un fichero CSS (b), o como regla directa dentro de nuestra hoja de estilos (c).

```
(a) <link rel="stylesheet" media="screen and (max-width: 480px)" href="extra.css" />
(b) @import url(extra.css) screen and (max-width: 480px);
(c) @media screen and (max-width: 480px) { ... }
```

En todos los casos lo que se hace es aplicar las reglas que incluyen, ya sea como fichero en los casos a y b o como estilos directamente en el caso c, de manera que tienen efecto solamente bajo las condiciones especificadas en la/s condición/es.

La diferencia entre *-width y *-device-width es sutil, pero importante... *-width hace referencia al ancho de la ventana, mientras que *-device-width al ancho de la pantalla, por lo que, aunque en un dispositivo móvil esto no tiene efecto al no poder redimensionar la ventana (a día de hoy), en el escritorio sí que es una diferencia importante y, por lo tanto, se recomienda no emplear *-device-width como atributo selector.

En el ejemplo existen dos elementos de la media query: el tipo y el atributo.

El tipo en el ejemplo es «screen», que define el escenario en el que se aplicaría, y el atributo sería «max-device-width», que en este caso establece un ancho máximo del dispositivo en 480 px. La combinación de ambas es la que permite o no aplicar los estilos que contiene.

Como vemos en el propio ejemplo, se pueden combinar varias reglas mediante «and» para lograr una regla más elaborada.

En RWD ha sido muy común hasta hace relativamente poco tiempo establecer un conjunto de puntos de corte (breakpoints) en determinadas medidas de ancho, como referencia para los cambios de layout. Si bien es una alternativa, hoy en día y sobre todo tras la aceptación como buen enfoque de la filosofía mobile first, existen puntos de vista que no defienden la existencia de tales puntos de corte de forma absoluta, sino que los relativizan en función de cada diseño / escenario, los difuminan evitando siempre emplear medidas absolutas en las estructuras mediante la aplicación de un layout completamente fluido, o defienden incluirlos en cada componente del diseño de forma separada, ya que todos los elementos de la web no tienen por qué seguir un mismo criterio en cuanto a espacios y layout.

Tomando como referencia el framework frontend más popular en la actualidad (Bootstrap en su versión 3), los puntos de corte que establece en su versión estándar son los siguientes:

```
/* Dispositivos especialmente pequeños (phones, menores que 768px) */
/* Estilos principales, ya que es el contexto por defecto en Bootstrap */
@media (min-width: 768px) {
/* Dispositivos pequeños (tablets, 768px y superiores) */
}
@media (min-width: 992px) {
/* Dispositivos medios (escritorio, 992px y superiores) */
}
@media (min-width: 1200px) {
/* Dispositivos grandes (escritorios grandes, 1200px y superiores) */
}
}
```

No obstante, hoy en día es muy común que los frameworks frontend permitan al usuario personalizar estas características, precisamente asumiendo que, efectivamente, los puntos de corte generalmente no deben ser una decisión aislada del contexto del diseño, contenidos y de cada caso particular.

Para ver todos estos conceptos en la práctica, analizaremos un escenario muy habitual que se produce en RWD: la adaptación de un layout a columnas a un layout a columna única para los dispositivos más pequeños... o mejor dicho, siguiendo la filosofía mobile first, pasar de un layout de una columna a ancho completo a un layout de múltiples columnas. Veamos este ejemplo con código:

Posible marcado HTML

```
[...]
<meta name="viewport" content="width=device-width, initial-scale=1">
[...]
<div id="destacados">
        <article>
                <h2>Primer destacado</h2>
                <p>Lorem ipsum dolor sit amet... veniam quis.</p>
                <p><a href="#">Más información</a></p>
        </article>
        <article>
                <h2>Segundo destacado</h2>
                <p>Lorem ipsum dolor sit amet... veniam quis.</p>
                <p><a href="#">Más información</a></p>3
        </article>
        <article>
                <h2>Tercer destacado</h2>
                <p>Lorem ipsum dolor sit amet... veniam quis.</p>
                <p><a href="#">Más información</a></p>
        </article>
        <br class="clear"/>
</div>
[...]
```

Como podemos ver, se trata de 3 elementos englobados por etiquetas «article» que en su interior tienen un titular, un párrafo y un enlace de más información. Por defecto, y por la naturaleza de las etiquetas HTML empleadas, cada «article» sale con un «width» del 100 % de su contenedor, y por lo tanto, en este caso a una columna de ancho completo, con un padding de 20 px por cada lado.

Gracias al uso de del modelo de caja «border-box» que veremos más adelante, logramos poder trabajar con anchos relativos (fundamental en el diseño líquido con RWD) aunque empleemos paddings en medidas absolutas.

Hoja de estilos CSS

```
* {
        box-sizing: border-box;
        -moz-box-sizing: border-box;
        -webkit-box-sizing: border-box;
}
[...]
article {
        padding: 20px;
}
@media (min-width: 481px) {

        #destacados > article {

                float: left;
                width: 50%;
        }
}
@media (min-width: 768px) {

        #destacados > article {
                width: 33%;
        }
}
```

Por lo tanto, en la hoja de estilos podemos ver que sin media queries lo único que hacemos es dar ese padding a los «articles» (pensando en el contexto móvil más restrictivo) para pasar entre 481 px y 767 px a un formato de 2 columnas, y finalmente, de 768 px en adelante a 3 columnas.

Gracias a la aproximación mobile first vemos cómo vamos ajustando los layouts en contextos menos restrictivos, y que en la segunda media query ni siquiera tenemos que definir el «float», porque se ha aplicado en la anterior, y simplemente lo que hacemos es extender o sobreescribir la propiedad «width». Algunos autores reconocidos incluso aplican estos criterios relativos en la definición de los puntos de corte con las propias media queries, de manera que en vez de especificarlos en «px», lo hacen en «em».

Obviamente, este ejemplo simplemente pretende explicar la mecánica que se establece cuando se trabaja con RWD, combinando etiquetado HTML y estilos CSS, pero no quiere dar a entender que los puntos de corte establecidos son los que hay que aplicar.

7.3. Modelo de caja: border-box

Como decíamos, el RWD no debería aplicar layouts fijos en cuanto a estructuras en diferentes contextos, sino que entre los diferentes puntos de corte debería tener suficiente flexibilidad como para continuar con ese criterio de adaptación, para lo cual, se hace imprescindible el uso de valores porcentuales o mediante medidas relativas (em por ejemplo) para definir los anchos de las estructuras.

Esta característica ha hecho que se haya popularizado el uso del modelo de caja aplicado mediante la propiedad CSS «box-sizing» (a ampliar con prefijos -mox- y -webkit-) y su valor «border-box», que hoy en día está muy bien soportado por los navegadores.

Este modelo de caja hace que la propiedad «width» incluya el «padding» y el «border», de manera que los valores de estos se descuenten al total, para que el «width» sea siempre el que manda con su valor, independientemente de los cambios de «padding» y «border».

7.4. Sistemas de rejilla

Siguiendo las consideraciones que hemos realizado tanto en el ejemplo de aplicación de media queries como las del modelo de caja, nos toca hablar ahora de los sistemas de rejilla (grid systems).

La experiencia y análisis de la aplicación del RWD a proyectos web ha llevado a muchos desarrolladores a constatar cierto comportamiento deseable y habitual a la hora de gestionar la adaptación de las principales estructuras de contenidos de una web, que ha propiciado que existan en la actualidad multitud de estos sistemas de rejilla.

La base de estas rejillas es el ajuste de los contenidos alrededor de un sistema de filas que contienen columnas que dividen el espacio horizontal disponible, de manera que se puedan trabajar los cambios en los diferentes contextos contemplados en términos de columna en lugar de en medidas concretas. Además, estos sistemas de columnas se han terminado creando con un comportamiento fluido, por lo que no solamente los elementos pueden reajustar su estructura a diferentes columnas en cada contexto, sino que pueden acompañar de forma fluida los posibles escenarios intermedios entre puntos de corte. En la actualidad existen incluso herramientas que nos facilitan la tarea a la hora de crear estas rejillas responsive.

Estos sistemas de rejilla son prácticamente la base de todos los frameworks frontend que aplican la técnica de RWD, de manera que analizando el más extendido de ellos –Bootstrap– se puede encontrar una rejilla por defecto de 12 columnas, que permite gestionar no solamente los cambios de tamaño de los

contenidos en sus cuatro contextos responsive (xs, sm, md y lg), sino aspectos como la ordenación (push y pull), visibilidad (visible y hidden) y espaciado entre ellos (offset).

Precisamente en estructuras como estas rejillas flexibles y fluidas es donde el modelo de caja visto con anterioridad cobra más sentido, ya que las columnas que los conforman pueden tener un width porcentual (sin «margin») y jugar tanto como se quiera con el padding o el borde sin que afecte al conjunto de las columnas... logrando por ejemplo tener un padding en «px» en una columna con un ancho en «%».

7.5. Texto / Tipografía

Uno de los elementos más relevantes, y que más puede sufrir en los diferentes contextos que trata de solventar el RWD, es el texto o tipografía, máxime en los tiempos actuales, en los que es posible emplear diferentes familias de fuentes, lo que aumenta la complejidad de la toma de decisiones al respecto.

Obviamente, existen importantes connotaciones en cómo gestionar el texto cuando creamos un proyecto que se ha de adaptar no solamente a diferentes tamaños de pantalla, sino densidades de píxel, posibles modificaciones establecidas por el usuario desde la configuración de su navegador, diferentes distancias de lectura (móvil/tableta/escritorio), etc. Además, cuando indicamos «gestionar el texto», no solamente nos referimos a su tamaño, sino a otros aspectos como por ejemplo la altura de línea, que es un factor importante a la hora de lograr la legibilidad adecuada, etc.

Lógicamente, podemos emplear las ya conocidas media queries para aplicar diferentes estilos en función de los contextos que contemplemos, pero es interesante establecer algunas de las bases a tener en cuenta:

1) Los diferentes dispositivos pueden establecer (y de hecho establecen) tamaños base diferentes para la tipografía.

2) Un tamaño de 16 px es aceptado como un valor correcto para el cuerpo del documento, y por debajo de dicho valor empieza a considerarse no apropiado.

3) Diferentes tipografías tienen comportamientos diferentes en los navegadores, e incluso la misma tipografía puede variar notablemente al renderizarse en ellos (para lo que existen interesantes herramientas de testeo [1 y 2]), por lo que la elección de la fuente es un paso crucial para su legibilidad.

4) Se ha de considerar el texto como parte de la interfaz de usuario, no como un elemento meramente decorativo o sin función.

5) Es recomendable evitar el uso de medidas absolutas (px), y emplear medidas que permitan gestionar correctamente tamaños relativos o cambios marcados por el usuario, como em o rem. El uso de estas medidas nos permitirá ajustar el tamaño de todas las fuentes del proyecto simplemente ajustando en cada media query el tamaño base del elemento html.

6) Las pantallas de dispositivos reducen el tamaño del texto debido a su densidad de píxel. Por inercia se tiende a pensar por tanto que en dispositivos móviles se ha de aumentar la fuente, lo cual no es necesariamente cierto si la fuente base es suficientemente grande (ver punto 2), ya que la distancia de lectura en estos dispositivos es mucho menor y además aumenta notablemente la nitidez de la misma.

7) Para algunos casos puede ser interesante aplicar la propiedad «word-wrap» con el valor «break-word», de manera que nos aseguremos de que el contenido no sobrepasa los límites deseados.

Para determinadas situaciones en las que necesitamos que el texto sea el que se adapte a su contenedor, existen interesantes librerías en JavaScript, que permiten este ajuste: FlowType.js, que permite la gestión de texto que fluye en un contenedor con un tamaño de fuente basado en el ancho de dicho contenedor, o FitText.js, BigText.js y SlabText.js para la adaptación de titulares.

Por último, no podíamos dejar de comentar el uso de las tipografías conocidas como icon-fonts, que nos permiten emplear iconos tratados como texto, lo cual tiene de forma directa numerosas ventajas:

- Renderizado óptimo en pantallas con diferentes densidades de píxel.
- Facilidad para cambiar sus propiedades, aplicando CSS convencional como a cualquier texto (color, efectos, etc.).
- Correspondencia en tamaños con textos colindantes.
- Sencillez de uso y de mantenimiento.
- Optimización de peticiones HTTP al funcionar al estilo de una spritesheet.

Si bien es cierto que suelen requerir un peso algo mayor como archivo que el uso individual de los iconos necesarios, hoy en día esto puede solventarse gracias a herramientas que nos permiten crear nuestra propia tipografía de iconos, solamente con los necesarios ya sean propios o de entre algunas fuentes libres disponibles (1, 2 y 3).

7.6. Imágenes

Otros elementos principales que todo proyecto web incluye hoy en día y que deben adaptarse en un proceso RWD son las imágenes. Al igual que con los textos, la necesidad de abordar principalmente diferentes tamaños (por cambios de layout o por la propia fluidez del contenido entre puntos de corte) y diferentes densidades de pantallas hace que estos elementos requieran una forma de adaptación.

Lo primero que tenemos que diferenciar son las imágenes que se pueden especificar por CSS, como por ejemplo, los fondos, de las imágenes que son puro contenido en el documento (etiqueta <img>).

Para las primeras, el uso convencional de las media queries nos permite dar una solución bastante correcta (unido a las capacidades de comportamiento a través de la propiedad background-size), pero para las segundas, la solución que marca la base del comportamiento con CSS

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

deja al descubierto algunos problemas importantes:

- Peso / consumo de ancho de banda, ya que en muchos escenarios se descarga una imagen mayor a la estrictamente requerida.

- En ocasiones no se desea solamente un reescalado, sino cambiar la porción de la imagen mostrada (técnica denominada «art-direction»).

Por tanto, cuando se habla de imágenes en RWD, se hace alusión a la capacidad de intercambiar los orígenes «src» de las imágenes más que al hecho en sí de modificar su apariencia en función del espacio disponible, que con un código CSS similar al anterior tendríamos prácticamente solucionado.

El uso de icon fonts (como se ha visto en el anterior punto) y el uso correcto del formato SVG SVG nos permite solucionar este problema, pero desgraciadamente solo en alguna tipología concreta de imagen.

Existen infinidad de técnicas y bibliotecas que tratan de dar solución a esta problemática (incluso la propia propuesta del W3C basada en el elemento <picture>, que funciona como contenedor para la imagen y permite disponer de varias fuentes y tomar la adecuada en función de una media query CSS).

Entre todas ellas destaca actualmente un polyfill denominado *picturefill*, que emula el futuro comportamiento del estándar <picture> mediante etiquetas soportadas ampliamente por los navegadores actuales.

Dentro de las técnicas de lado de servidor, existen soluciones como *adaptive-images* basadas en el uso de scripts en PHP y htaccess, otras basadas en la generación al vuelo con cacheado como Smart Image Resizer a partir del propio «src» de la imagen, e incluso servicios en la nube libres de pago (1 y 2), que permiten la generación de imágenes bajo demanda.

Por otro lado y como era de esperar, en JavaScript existen también multitud de scripts que permiten afrontar esta problemática con mejor o peor resultado, la mayoría apoyándose en atributos «data-*», entre los que destacan:

- rwd.images.js (con fallback a noscript y soporte para art-direction)
- hisrc (plugin de jQuery)
- foresight.js

8. Elementos especialmente problemáticos en RWD

Además de las imágenes aplicadas mediante la etiqueta <img> que acabamos de ver, existen una serie de elementos especialmente problemáticos cuando se desarrolla con RWD.

8.1. Contenido embebido mediante iframe

Contenido como vídeos o mapas, que se embeben en la web a través de un iframe, resulta especialmente comprometido a la hora de enfocar su comportamiento RWD. Una técnica que da buenos resultados para solventar esta situación es la siguiente:

Englobar el iframe en una etiqueta «div» que haga de contenedor.

Aplicar estos estilos al contenedor:

```
.div-container {
        position: relative;
        padding-bottom: 75%;
        height: 0;
        overflow: hidden;
}
```

Donde el valor del padding-bottom viene determinado por la relación del ancho y alto del iframe original (100 * alto/ancho), lo que por ejemplo para un iframe con aspect ratio de 4:3 nos da precisamente 75 = 100 * ¾.

Aplicar estos estilos al iframe:

```
.div-container > iframe {
        position: absolute;
       top: 0;
       left: 0;
       width: 100%;
       height: 100%;
}
```

Finalmente, es posible que para un contenido en iframe como pueda ser por ejemplo un vídeo de YouTube, se tenga que aplicar un «padding-top» específico, para, como en ese caso, permitir espacio para la barra superior que el visor añade al vídeo, quedando por tanto la modificación para un vídeo de YouTube de este modo:

```
.youtube-container {
        position: relative;
       padding-top: 35px;
        padding-bottom: 75%;
        height: 0;
        overflow: hidden;
}
```

Por otro lado, si deseamos que el iframe solo se adapte en ancho, manteniendo su alto fijo, un CSS simple que podría lograrlo es el siguiente:

```
#my-iframe { max-width: 100%; }
```

8.2. Tablas

Por la propia naturaleza de los datos tabulares, su adaptación a diferentes escenarios o contextos como los que asume el RWD es una tarea compleja que en la actualidad se ha afrontado con diferentes técnicas reconocidas:

1) Ir ocultando columnas y quedándonos con las más relevantes, según vamos reduciendo el ancho de la tabla.

2) Intercambiar las filas por las columnas para lograr tener solamente un scroll horizontal.

3) Convertir la tabla a una versión sencilla agrupando los valores de cada registro (dejando por tanto 2 columnas y mucha más longitud vertical, que se entiende no tan problemática en un contexto móvil).

4) Fijar una columna como referencia y aplicar un scroll interno al resto de las columnas (y su demo).

5) Mostrar una gráfica a partir de un ancho mínimo si dicha gráfica permite representar la información relevante.

6) Disponer de una imagen representativa que sustituye a la tabla en determinados tamaños mínimos, y que enlaza con la tabla de forma independiente para poder visualizarla con scroll.

Existe alguna buena comparativa con enlaces a sus referencias sobre todas estas posibles soluciones, y en el caso de las tres primeras opciones, existen diferentes soluciones que están perfectamente detalladas e implementadas en un recurso que las recoge de forma conjunta, y también existen algunas soluciones más elaboradas (1 y 2) que combinan algunas de las opciones que engloban otras soluciones.

Finalmente, una técnica muy sencilla que permite generar scroll cuando la tabla ya no puede «comprimirse más en ancho» es envolverla en un contenedor que genere scroll cuando sea necesario, con un código similar a este:

```
[...]
.overflow-container {
overflow-y: scroll;
}

[...]

<div class="overflow-container">

 <table>
   <!-table contents ->
</table>
</div>
```

8.3. Texto preformateado

El texto preformateado no se ajusta a su contenedor (el conocido «word-wrap») por defecto, por lo que es un elemento conflictivo para el RWD. Una de las técnicas CSS más ampliamente aceptadas para lograr dar una solución a este comportamiento es sobreescribir el comportamiento CSS de la etiqueta «pre»:

```
pre {
 white-space: pre-wrap;       /* css-3 */
 white-space: -moz-pre-wrap;  /* Mozilla, since 1999 */
 white-space: -pre-wrap;      /* Opera 4-6 */
 white-space: -o-pre-wrap;    /* Opera 7 */
 word-wrap: break-word;       /* Internet Explorer 5.5+ */
}
```

La otra alternativa es envolver la etiqueta en un contenedor que disponga de scroll horizontal, al igual que se ha podido ver en el último ejemplo para el caso de las tablas.

8.4. Canvas

En el caso del elemento canvas, la forma de lograr un comportamiento adecuado en RWD de una forma relativamente sencilla es dando valor a su ancho y alto en el etiquetado HTML, y aplicando posteriormente un valor CSS a su propiedad «width» del 100 %, de manera que el alto se adapte automáticamente de forma proporcional.

```
[...]#my-canvas {
```

```
width: 100%;
height: auto;
}
[...]
<canvas id="my-canvas" width="800" height="600" />
```

Obviamente, con este etiquetado el ajuste será con respecto al elemento que lo contiene, que puede marcar a su vez un ancho variable, pero lo hará proporcionalmente porque parte del tamaño original correcto a una escala del 100 %, especificado mediante los atributos «width» y «height» de la etiqueta «canvas».

8.5. RWD para email

Si bien la inmensa mayoría de los esfuerzos en los últimos años se han centrado en la web, cada vez es más necesario solventar estos mismos contextos en un escenario que de por sí suele ser aún más problemático: los clientes de correo electrónico.

Por suerte, hoy en día ya existen algunas opciones que nos permiten no partir desde cero, y lidiar con la gran cantidad de restricciones y problemas que genera el renderizado HTML dentro de los gestores de correo, ya sean aplicaciones o clientes web. Entre las más destacadas encontramos:

- Ink (de los creadores del framework Foundation)
- Zurb responsive email templates
- Plantillas de la conocida herramienta de email marketing: MailChimp
- Antwort [75]

9. Frameworks más relevantes

Prácticamente todos los frameworks frontend hoy en día abrazan el RWD como técnica de desarrollo web, partiendo como hemos dicho de un sistema flexible y fluido de rejilla.

Dentro de estos frameworks podríamos catalogar los que son estrictamente estructurales y centrados en dicha rejilla, y los que tienen un carácter más completo y que habitualmente engloban estilos adicionales, utilidades, componentes e incluso JavaScript.

De entre los principales existentes, destacamos 3 como referentes o bien por su potencia y reconocimiento, o bien por su sencillez:

9.1. Bootstrap 3

http://getbootstrap.com/

El framework frontend completo más popular en la actualidad: mobile first en esta última versión, RWD, con soporte para LESS y SASS, personalizable y complementado con un gran conjunto de estilos adicionales, icon-font, componentes, comportamientos en JavaScript…

9.2. Foundation 5

http://foundation.zurb.com/

Otro de los frameworks frontend completos mejor valorados por los desarrolladores: filosofía mobile first, RWD, con soporte para SASS, personalizable y complementado con plantillas, iconos, componentes, JavaScript...

9.3. Unsemantic

http://unsemantic.com/

La versión sucesora con RWD del conocidísimo grid960, uno de los sistemas de rejilla más interesantes para comprender su funcionamiento y aislarse de todo lo que no es estrictamente necesario.