POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID

Educación para vivir mejor

Construcción de elementos de software web 1

Semana 6

Diseño Responsivo: Flex - Grid

Media Técnica en Programación de Sistemas de Información
Politécnico Jaime Isaza Cadavid - 2020

¿Qué es diseño flexible?

Antes de analizar y evidenciar cómo es la maquetación responsive vamos a mirar a lo que podría ser un antecesor: el diseño flexible, o líquido.

Se dice que un diseño es líquido, o más correctamente, que una maquetación es líquida cuando definimos todos los anchos de los contenedores en términos de porcentajes de la vista del navegador, así pueden expandirse y contraerse cuando la ventana del navegador cambie de tamaño.

Muchas páginas web eran maquetadas así, cuando el tema de las pantallas y dispositivos no era todavía un problema. Cuando empezaron a salir las primeras pantallas diferentes **(800x600 y 1024x768),** está maquetación por porcentajes te permitía tener ese margen entre unas dimensiones y otras.

Con el paso del tiempo, hemos visto que el problema ya no se resuelve teniendo algo de margen ya que a día de hoy tenemos pantallas que van desde los **320px** de ancho hasta los **2560x1440**. Aquí es donde entra el diseño responsive.

¿Qué es diseño responsive?

El responsive, o Responsive Web Design (RWD) es un concepto de desarrollo orientado a que los sitios web se vean y comporten correctamente en todos los dispositivos y pantallas.

Responsive no es solo móvil, tablet y escritorio, también es un navegador que ocupe la mitad de la pantalla por alguna razón. Por ejemplo, estamos escribiendo un artículo en nuestro editor de texto y tenemos al lado una web donde consultamos información.

Esto se consigue combinando varios enfoques:

- Maquetar ciertas zonas usando porcentajes.
- Ciertos elementos se ajustarán al tamaño del contenedor hasta un cierto valor, es decir, se comportan como si tuviesen las medidas por porcentajes pero pudiendo marcar tamaños máximos y mínimos.

- Usaremos unidades "flexibles" en ciertos casos, como rem o vw y vh. Y claro, el %
- Tenemos a nuestra disposición unas expresiones CSS, las **mediaqueries**, que nos permiten aplicar una serie de reglas cuando en nuestro navegador se cumplen unas condiciones: ancho, alto, resolución, orientación.
- Veremos qué es el viewport y cómo usarlo.

Imágenes, media y tipografía flexibles

Imágenes

En este entorno Responsive las imágenes y videos tienen su forma de afrontarlos al implementarlos en nuestros productos web y aunque ambos formatos podemos dimensionarlos por porcentajes tenemos varios casos en los que la imagen o el vídeo queda demasiado grande o demasiado pequeño. Para ello tenemos unas propiedades que matizan el width (ancho) o el height (alto):

- max-width
- min-width
- max-height
- min-height

De esta manera no solo podemos definir un ancho/alto en porcentajes sino marcar unos límites.

Por ejemplo: Quiero que mi imagen se ajuste al 100% del contenedor pero solo hasta un máximo de 600px, controlando de esa manera que se haga demasiado grande.
En maquetación responsive es súper buena idea meter las imágenes de contenido en un contenedor, dar a la imagen un ancho del 100% y usar el contenedor para definir el tamaño que debe tener la imagen. Esto se hace para poder controlar diferentes imágenes cuando no podemos asegurar los tamaños que van a tener sin que nos interfiera con la estructura.

Videos

Los vídeos son más complicados de controlar porque la etiqueta **<video>**, al contrario que la **<img>** no redimensiona proporcionalmente el video, aunque hay formas de suplirlo ya sea por CSS o haciendo uso de unas librerías en JavaScript.

Tipografía

Para el tema de la tipografía hay varias escuelas y formas de afrontarlo según el caso:

- Una es usar unidades fijas como los píxeles, e indicar en cada caso qué tamaño de fuente debe tener cada texto. Por ejemplo: en móvil mi texto básico será de 18px porque la pantalla es más pequeña y quiero que se lea mejor, pero en tablet lo bajo a 16px y en escritorio lo vuelvo a poner a 18px. Aunque en pantallas muy grandes usaré 20px o 24px.
- Tenemos unas unidades relativas como los **rem**

rem: es una unidad relativa al tamaño de texto especificado en el elemento raíz de nuestro documento que es el **<html>**. De manera que si el elemento **html** está a 16px (valor por defecto) **1rem** equivaldrá a esos **16px.**

Esto nos ayuda a poder ajustar todas nuestras medidas de forma proporcional solo cambiando el tamaño de fuente de la etiqueta <html>.

Media Queries

Las **mediaqueries** son las instrucciones que nos permiten aplicar una serie de reglas CSS cuando se cumplan una o varias condiciones. Tienen este aspecto:

```
@media all and (min-width:500px) {
    /* Reglas CSS que aplicaremos */
}
```

Viewport

Aquí es donde entra en juego una etiqueta que la conocíamos por otros motivos pero que ahora tiene un contenido especial: el viewport.

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

Los dispositivos móviles (teléfonos y tablets) utilizan este "viewport" para mostrar la página de una forma curiosa: salvo que se le indique lo contrario intentarán mostrarla al máximo tamaño posible, haciendo suficiente zoom para permitir que las páginas no adaptadas se puedan ver (aunque diminutas).

Con la etiqueta meta viewport le decimos a estos dispositivos cómo tienen que comportarse. El ejemplo anterior es el más típico y le dice al viewport de un dispositivo móvil que:

1) el ancho del viewport debe coincidir con el ancho del dispositivo.
2) la escala inicial siempre será el 100%.

Mobile first

Hay varias formas de afrontar la maquetación de un proyecto responsive, la que nos parece más adecuada es mobile first donde se plantean primero los estilos de la vista de móvil y conforme tenemos mayor tamaño de pantalla sobreescribimos o ajustamos los estilos necesarios.

La maquetación mobile first da prioridad a los dispositivos con menos capacidad de pantalla, conexión y batería de manera que, por ejemplo, se cargan imágenes más pequeñas que consumen muchos menos datos o se muestra una maquetación más simple y adaptada a que usemos los dedos como punteros (botones y zonas clicables más grandes). Y si luego resulta que estamos viendo la página en un una pantalla más grande, gracias a las **mediaqueries**, podemos cargar elementos más pesados, mostrar otros que en móvil pueden no tener sentido y adaptarlo todo a que ya se va a manejar con ratón y/o teclado.

Un ejemplo sería la típica página que tiene una primera sección con una imagen de fondo enorme: con la maquetación mobile first colocaremos primero un fondo adaptado a una pantalla más pequeña con el consiguiente ahorro en nuestra tarifa de datos. Si alguien la carga desde escritorio, donde suele tener una conexión con tarifa plana, pasaremos a mostrar una imagen de fondo mucho más grande, acorde a la pantalla que esté usando. Pero nunca obligamos a quien va en el metro a descargarse una imagen descomunal de ochocientos **megapíxels** en su móvil con una tarifa de datos que se paga con sangre.

En el CSS esto se representa escribiendo primero los CSS que se verán en las pantallas/ventanas de navegador más pequeñas. Posteriormente añadiremos, dentro de mediaqueries, los ajustes necesarios para los tamaños mayores de pantalla o ventana.

NOTA: Recordemos que responsive no es solo tema de dispositivos. Claro que hay móviles de 320, tablets de 768 y pantallas desde 1336 a 2560 de ancho. Pero responsive también es un tipo en una pantalla más modesta, por ejemplo cuando una usuaria pone el navegador en la mitad de la pantalla mientras trabaja y en la otra mitad poner un reproductor de vídeo porque está enganchada a Juego de Tronos.

Entonces, ¿cuántos breakpoints hay? ¿hay unos más estándar que otros?

Realmente hay una serie de anchos que se tienen más en cuenta que otros. por ejemplo, un tablet suele tener de 768px (en vertical) a 1024px (en horizontal). En ordenadores de escritorio hay una serie de anchos que se suelen repetirse en los diseños (1280, 1500, 1600).

Todo esto es muy relativo pero cuando el equipo de diseño entrega un diseño suele hacer de 2 a 3/4 vistas, y tomaremos esas medidas como nuestros breakpoints principales. El objetivo es clavar el diseño en esos puntos.

Luego tendremos breakpoints menores que nos los da el contenido: desde el tamaño más pequeño vamos redimensionando el navegador y cuando algo "se rompe" (o se descoloca), creamos un breakpoint y arreglamos lo que se haya roto o desencajado.

De esta manera nos aseguramos de que en cualquier ventana de navegador nuestra página se va a ver correcta. Quizás no se vea de una manera ideal, pero no se va a ver descolocada.

Flexbox

La propiedad Flexible Box, o flexbox, de CSS3 es un modo de diseño que permite colocar los elementos de una página para que se comporten de forma predecible cuando el diseño de la página debe acomodarse a diferentes tamaños de pantalla y diferentes dispositivos.

Hasta que apareció Flexbox la única manera de distribuir elementos en horizontal (por ejemplo: un menú horizontal, o un bloque principal y uno secundario) era calculando el espacio que debe ocupar cada bloque y controlando mucho que ningún elemento se saliese de lo planificado.

Con flexbox podemos disponer de un bloque flexible que de manera natural se ajuste dentro de unos parámetros que le indiquemos.

Flexbox es una herramienta imprescindible en la maquetación actual y nos permite tener elementos HTML que se ajusten a las diferentes pantallas de los diferentes dispositivos.

En qué casos se utiliza FlexBox

Pues hay casos muy evidentes, como hacer un pie fijo cuando hay poco contenido, o un panel lateral con una parte fija y otra flexible. Pero también hay otros más simples como un menú horizontal, un listado de iconos de redes sociales o una noticia donde queremos que a veces la imagen vaya arriba o tras el texto. Básicamente lo podemos/querremos usar en cualquier estructura que se distribuya en vertical u horizontal y sobre la que queramos controlar el espaciado, orden o alineamiento.

Mira esta guía completa: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Grid

Un sistema de grid o rejilla, nos permite disponer los elementos de una página y que estén alineados. Va a existir una rejilla imaginaria de filas y columnas, a partir de la cual vamos a colocar los elementos de nuestra web. El uso de un sistema de grid tiene sentido si acompaña un diseño que usa también una rejilla.

Un sistema de grid nos sirve para posicionar los elementos de la página alineados. Se usa en un montón de webs, por ejemplo, este ejemplo de Google.

Grid Google Plus

Podemos ver que los elementos están dispuestos en filas y columnas. Hay 4 columnas que se ven claramente, con un elemento que se expande en 2. Aunque a simple vista parecen no estar alineados en filas, todas las cajas tienen una altura proporcional a una base.

CSS grid

CSS grid es una nueva característica de CSS que permite tener un sistema de grid de forma nativa en CSS. Es una herramienta compleja, así que vamos a ver las bases de cómo poder usarla.

En primer lugar, existen 2 tipos de elementos, el contenedor del grid y los elementos del grid. En este sentido, es similar a algo que ya conocemos: flexbox.

Para comenzar, usaremos en el contenedor la propiedad display:grid y definiremos las filas y columnas de nuestro grid con grid-template.

```
.wrapper{
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 40px 200px 40px;
}
```

En este grid vamos a tener 4 columnas, cada una de tamaño **1fr**, que es una medida sobre el espacio disponible (free space). Por tanto, se divide el espacio disponible en 4 partes para las columnas. Para las filas, tendremos 3 de 40, 200 y 40px respectivamente.

A continuación, indicaremos a los elementos si queremos que ocupen una o varias filas o columnas con las propiedades **grid-column y grid-row.**

```
.item1 {
    grid-column-start: 1;
    grid-column-end: 4;
}
```

Esto indica que se expande desde la primera línea de grid hasta la cuarta, es decir, ocupa las 3 primeras columnas.

NOTA: si tenemos 4 columnas, tendremos 5 líneas de grid. Es decir, siempre vamos a tener una más que el número de filas o columnas.

Podemos escribir lo anterior de una forma simplificada:

```
.item1 {
    grid-column: 1 / 4;
}
```

Para las filas funciona exactamente igual:

```
.item3 {
    grid-row-start: 2;
    grid-row-end: 4;
}
```

También podemos indicar el tamaño del espaciado de elementos en el contenedor con la propiedad **grid-gap.** Podemos indicar 2 valores si queremos espaciado distinto entre filas y columnas.

Veamos un ejemplo:

HTML

```
<div class="wrapper">
  <div class="item item1">1</div>
  <div class="item item2">2</div>
  <div class="item item3">3</div>
  <div class="item item4">4</div>
  <div class="item item5">5</div>
  <div class="item item6">6</div>
</div>
```

CSS (SCSS)

```
body{margin:0;}
.wrapper{
  height: 100vh;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 5px;
}

.item{
  background: deeppink;
  color: white;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

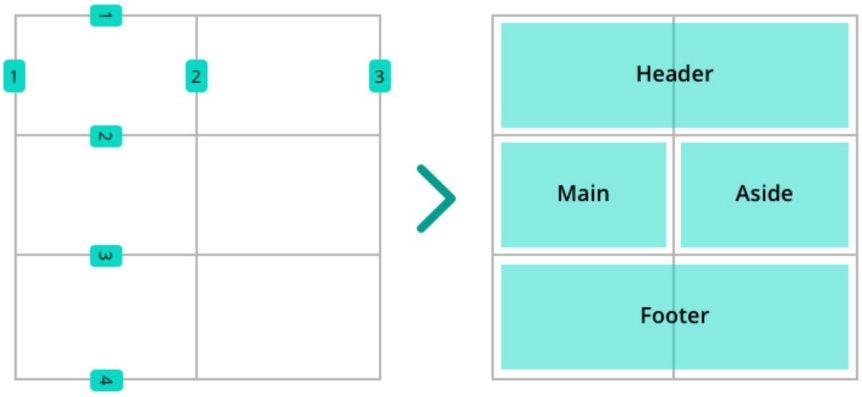
¿Cuál es el resultado al escribir el código HTML y CSS?

A la hora de posicionar los elementos en el grid, también podemos usar la palabra **span** para indicar cuánto se expande desde la fila/columna actual. Por ejemplo, para el item1 que se expandía desde la línea de grid 1 a la 4 podríamos usar:

```
.item1 {
    grid-column: span 3;
}
```

Pero tenemos una forma de decirle a nuestro grid cómo comportarse cuando haya más elementos de la cuenta, gracias a **grid-auto-rows y grid-auto-columns**, funcionan como **grid-template-rows/columns** pero solo se aplica en el caso de que haya más elementos de la cuenta.

Y lo podemos usar junto con **grid-auto-flow**, que fuerza una única dirección (columna o fila) para nuestra rejilla.



El grid que hay debajo es de 2x3, y para definirlo usamos:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
}
```

pero si queremos "nombrar los espacios" podemos usar, además, **grid-template-areas:**

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-areas: "header header" "main aside" "footer footer"
}
```

De esta manera creamos un área completa asignando a dos el mismo nombre, como el header o el footer.

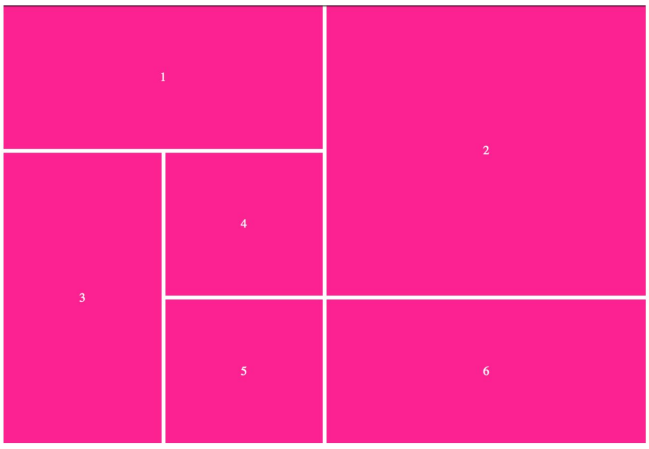
Si ahora quisiéramos que un elemento en concreto se colocará en una de estas áreas, solo tendríamos que decírselo:

```
.item--1 {
  grid-area: main;
}
```

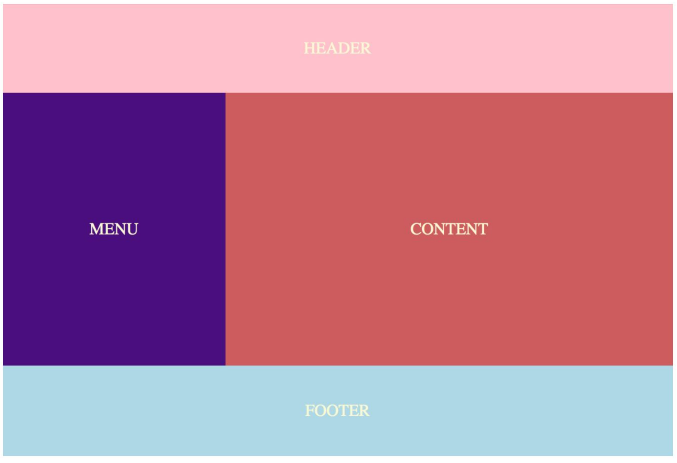
Taller: Ejercicios Práctico

1. **Realizar una página web tributo.**
    a. **Ejemplo: https://codepen.io/freeCodeCamp/full/zNqgVx**

2. **Realice el siguiente ejemplo haciendo uso GRID**



3. **Vamos a crear un grid de 12 columnas y 3 filas, la primera y la última ocupan el 20% del alto del viewport. Crearemos la composición de la imagen.**



Quiero saber más ...

Sobre Diseño Responsivo

- https://www.youtube.com/playlist?list=PLQCgNGUqLK4mW7LxW3jJdRjCnErL5rszl
- https://www.youtube.com/watch?v=y6zYUe7MdLQ
- https://www.youtube.com/watch?v=SEXm5OM-U3s
- https://www.youtube.com/watch?v=KjHRa_Qzus8&index=8&list=PL6hPvfzEEMDaKYAabXoDL7A-fZcwxvlqe

Sobre Flex

- Teoría para aprender FlexBox: https://www.youtube.com/watch?v=F-KCncXMPk0
- Juegos par aprender FlexBox
    - http://flexboxfroggy.com/#es
    - http://www.flexboxdefense.com/
- Página interactiva para aprender FlexBox: https://codepen.io/enxaneta/full/adLPwv/
- Guía completa sobre FlexBox: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Sobre Grid Layout

- Pensando en GRIDS: https://vimeo.com/98141102
- Aprendiendo CSS Grid: https://learncssgrid.com/
- Juego para aprender Grid Layout: https://cssgridgarden.com/#es