Diseños Responsivos (Parte II)

Orden y breakpoint de *media queries*

Antes de seguir implementando los *media queries* en nuestro proyecto, es importante tener en consideración dos conceptos muy importantes a la hora de construir diseños responsivos:

1. Orden de los Media Queries
2. Breakpoint

Orden de los *media queries*

Cuando estructuramos *media queries* debemos tener en cuenta el orden que tienen estos en las hojas de estilo.

Por ejemplo, si definimos varios *media queries* con diferentes tamaños y de manera desordenada en una hoja de estilo, es probable que una persona que quiera mantener nuestros estilos termine desorientada y tarde más de lo debido al encontrar un problema.

Por lo general, para solventar este problema de orden tendremos a dispocisión dos formas muy usadas que nos ayudarán a ordenar nuestros *media queries*.

1. *Desktop first*

La primera técnica para organizar nuestros estilos utilizando *media queries* es construyendo los estilos pensando desde un comienzo los dispositivos de escritorio.

Los estilos en esta técnica se construyen usando una serie de *media queries* que especifican el ancho máximo y por lo general el diseño en dispositivos pequeños es simple.

```
/* Estilos */
@media screen and (max-width: 1024px) {
  // reglas CSS
}
@media screen and (max-width: 768px) {
  // Reglas CSS
}
```

El problema de esta perspectiva está relacionado al factor tecnológico, puesto que al usar anchos máximos limitamos la experiencia en algunos dispositivos que tienen un tamaño menor al especificado *media query*, lo cual es un error tendiendo en cuenta lo variable que son los tamaños y resoluciones en los dispositivos móviles.

2. *Mobile First*

Sin embargo, la falencia que tiene *Desktop First* la podemos solventar usando *Mobile First*. Este método utiliza los *media queries* para ajustar el diseño en resoluciones más grandes, especificando el ancho mínimo que tendrá el primer dispositivo a afectar.

```
/* Estilos */
@media screen and (min-width: 320px) {
  // reglas CSS
}
@media screen and (min-width: 768px) {
  // Reglas CSS
}
```

Desarrollar usando este método nos permitirá garantizar que los dispositivos móviles que no soportan *media queries* puedan visualizar correctamente la interfaz.

Nosotros podemos trabajar con cualquiera de estos dos métodos, pero por sus ventajas asociadas a su implementación y código *CSS* más legible elegiremos ordenar nuestro *CSS* usando ***Mobile First***.

Escogiendo *breakpoints*

Estos elementos mencionados anteriormente deberán contar con un punto de quiebre que defina el ancho en que se aplicará el estilo.

El concepto de *breakpoint* se refiere al punto donde el dispositivo aplica el bloque de código contenido dentro de un *media query*.

Como sabemos los *media queries* definen su tamaño usando un ancho - mínimo o máximo dependiendo del caso - definido dentro de los paréntesis. Este tamaño puede ser definido teniendo en cuenta la popularidad del dispositivo o usando algún patrón popular de *breakpoints*.

En el primer caso, podemos investigar cuáles son lo dispositivos más usados y en base a esa investigación definir los tamaños a usar para smartphone, tablets y finalmente desktop.

En el segundo caso, podremos usar tamaños usados en la industria para definir nuestros breakpoints.

Por lo general los *breakpoints* más usados no están estandarizados, de modo que para elegir uno debemos avocarnos a la familiaridad y comodidad.

En nuestro caso, por su popularidad y fácil implementación, usaremos los ***breakpoints* de *Bootstrap*** para hacer responsivo el diseños de la maqueta.

Los *Breakpoints* usados por *Bootstrap* son:

Dispositivo	Tamaño
Dispositivos pequeños	@media (min-width: 576px) { ... }
Dispositivos medianos	@media (min-width: 768px) { ... }
Dispositivos grandes	@media (min-width: 992px) { ... }
Dispositivos extra grandes	@media (min-width: 1200px) { ... }

Lectura complementaria

- Common Breakpoints for Media Queries - StackOverFlow
- Responsive Breakpoints - Bootstrap

Implementando *media queries* en proyecto

Ahora aplicaremos las *media queries* en nuestra maqueta con el fin de que esta tenga un comportamiento responsive.

Ahora, que ya tenemos definido dos de los conceptos más importantes a la hora de implementar *media queries*, estamos listos para transformar nuestra maqueta.

Segmentar estilos de maqueta

En primer lugar, para trabajar de mejor manera usando la metodología *mobile first* vamos dejar todos los estilos que ya estaban en la maqueta dentro de un *media query* extra grande a modo de mostrar estos solamente cuando el ancho del *viewport* sea de `1200px`.

En este contexto, los estilos que traspasaremos a este *media query* se encuentran en el directorio `layout`. Estos definen el diseño realizado en la maqueta ya que contienen todas las reglas de posicionamiento y dimensiones de la maqueta.

Comencemos por el parcial `_header`. En el agregaremos un *media query* que contendrá un tamaño maximo de `960px`. Luego, moveremos los estilos hacia su interior.

```
@media only screen and (max-width: 960px){
  /* Header Layout
   ================================================================================
*/
  .header {
    display: inline-block;
    width: 100vw;
  }
}
```

Ahora, haremos lo mismo pero con `_main`. Primero creemos el *media query* con ancho maximo de `960px` y luego, cortemos y peguemos los estilos dentro de el.

```
@media only screen and (max-width: 960px){
  /* Hero Section
   ================================================================================
*/
  .hero-section__featured-image {
    width: 100vw;
    @include relative-converter(height, em, 700, 16);
    background-image: url(../images/main-image.jpg);
    background-repeat: no-repeat;
    background-position: center;
    background-size: cover;
  }
  ...
}
```

Y finalmente, haremos por última vez este paso con el parcial `_footer`. Agreguemos el *media query* y luego movemos el contenido del parcial hacial el.

```
@media only screen and (max-width: 960px) {
/* Footer Layout
   ================================================================================
*/
  .footer {
    width: 100vw;
    padding-top: 1rem;
    padding-bottom: 1rem;
    position: relative;
    background-color: $sonic-silver;
  }
  ...
}
```

Cuando guardemos los cambios y recarguemos nuestro navegador, no veremos ningún cambio con el *viewport* a tamaño escritorio, pero si vamos disminuyendo los estilos, por debajo de los `960px`, veremos que los estilos se ven afectados a los elementos de nuestra maqueta.

Definir estilos principales fuera de *media queries*

En segundo lugar, debemos definir los estilos principales que afectan a nuestro *layout*, como por ejemplo los colores de las secciones o reglas de posicionamiento que sean necesarias en todo tipo de dispositivo. La idea es intentar repetir la menor cantidad de código posible en cada uno de los *breakpoints* que vayamos agregando.

Comencemos con `_header`. Este elemento no contiene una regla que sea necesaria para todos los dispositivos, por ahora.

En cuanto al parcial `_main`, el identificar los estilos a reutilizar podrá tomar un poco de tiempo, de modo que la forma más fácil es copiar lo estilos que se encuentran dentro del *media query* y luego, pegarlos al principio de la hoja de estilos. Cuando esté allí podremos ir seleccionando que estilos son importantes para que funcionen en todos los dispositivos.

```
/* Hero Section
 ================================================================================ */
.hero-section__featured-image {
  background-image: url(../images/main-image.jpg);
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}
...
@media only screen and (min-width: 1200px){...}
```

Cuando terminemos de seleccionar los estilos a reutilizar, deberemos ir eliminando los estilos repetidos que se encuentren dentro del *media query*, a modo de evitar repetir código.

```
@media only screen and (max-width: 960px){
  /* Hero Section
   ================================================================================
*/
  .hero-section__featured-image {
    width: 100vw;
    @include relative-converter(height, em, 700, 16);
```

```
  }

  .hero-section__container {
    width: 100vw;
  }

  ...
}
```

Si recargamos la página, podremos ver que los elementos ahora contienen estilos que se mantienen aún cuando cambiemos de tamaño o dispositivo.

Construir nuevos estilos usando representación visual

Con esto último estaremos preparados para comenzar a dar los estilos a cada uno de los tamaños seleccionados. Estos tamaños tienen un flujo que definimos cuando presentamos la propuesta al cliente, o sea, cuando dibujamos el *sketch*.

Entonces, para poder crear los estilos en estos tamaños es necesario revisar la distribución de los elementos de esa representación visual y en base a eso desarrollar los estilos para nuestro diseño responsivo.

Si recordamos nuestra maqueta tiene un ancho fijo de `1180px` que nos limita a ese tamaño para todos los dispositivos, pero si quitamos este valor el contenido pierde su distribución y por ende deja de tener sentido la interfaz.

Para solucionar este problema usaremos un media query que active el ancho del 100 viewport width desde cierta pantalla.

Comencemos ingresando al directorio `base` y luego, en el parcial `_base`.

Después, agregaremos un media query que incluirá un la directriz `@media`, un `only` para compatibilidad entre navegadores, un medio del tipo `screen` y finalmente, una expresión que active el *media query* a los `960px` usando un `max-width`.

```
@media only screen and (max-width: 960px) {
  // Regla CSS
}
```

Finalmente, peguemos dentro del *media query* la siguiente regla.

```
@media only screen and (max-width: 960px) {
  body {
    width: 100vw;
  }
}
```

Ahora, si guardamos lo hecho, vamos al inspector de elementos y reducimos el tamaño del navegador veremos que efectivamente el diseño se adapta al 100 por ciento del viewport width por debajo de los `960px`, pero si aumentamos el *viewport* a un tamaño mayor, nuestra maqueta quedará fija respecto al tamaño de la pantalla o viewport.

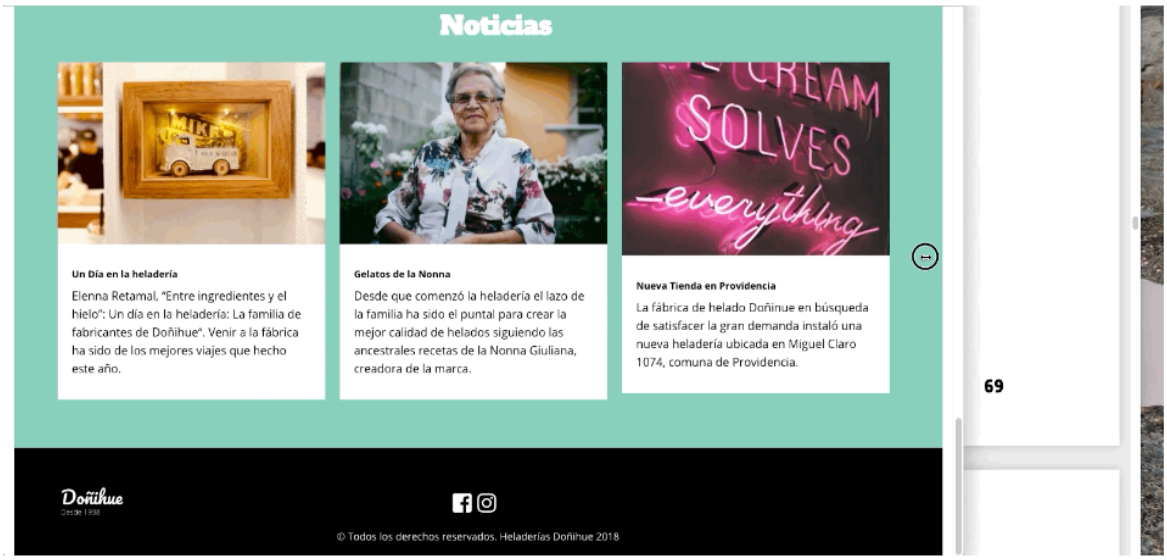

Imagen 1. Ejemplo de maqueta.

Con esto hemos conocido los conceptos principales a la hora de comenzar a trabajar un diseño responsivo usando *media queries*. Los estilos que faltaron por terminar en dispositivos móviles quedará como tarea para que desarrolles y practiques lo visto hasta ahora.

Ahora quedas con la misión de modificar el resto del sitio web, creando un flujo para dispositivos moviles.

Componentes de un diseño responsivo: Grillas

Anteriormente definimos como tarea el terminar de construir los estilos base para todos los dispositivos móviles.

Ahora conoceremos otro componente importarte a la hora de crear un diseño responsivo llamado grillas. Estas, como elemento de un diseño responsivo, no suponen ninguna novedad para nosotros ya que, la hemos usado muchas veces mientras construimos sitios web con *Bootstrap*, pero en realidad las grillas son mucho más que sólo un elemento importante de este *framework CSS*.

La importancia de las grillas

De hecho, esta técnica de crear grillas para ordenar elementos sobre un *"lienzo"* ha sido utilizado desde hace mucho tiempo antes de que se integrara a *Bootstrap*. Por ejemplo, en el siglo XX las grillas se usaban como base para estructurar diferentes piezas gráficas y de arte como *Poster y pinturas*. Ahora, el uso de estas en la web ha hecho que los diseños puedan ser elaborados y a su vez funcionen correctamente en diversos dispositivos.

Este último punto es importante, ya que las grillas cumplen una función primordial en el diseño responsivo, ya que su uso nos permitirá conocer cómo se moverá el flujo de los elementos y cómo se disponen los contenidos sin importar el tamaño del dispositivo a usar.

Opciones de grillas

Teniendo en cuenta lo anterior, el uso de una grilla en nuestro proyecto nos permitirá crear diseños responsivos de manera fácil, sin embargo, para poder hacerlo debemos conocer las opciones que tenemos para integrar una grilla.

Si nos ponemos a pensar sobre una grilla probablemente pensemos en *Bootstrap*, pero existen más opciones para implementar una grilla en nuestro proyecto.

Por ejemplo, podríamos usar una grilla de otro *framework CSS*, como la de *Zurb Foundation* o *Materialize* o simplemente crear una usando propiedades *CSS* como *floats*, *flexbox* o *CSS Grid*.

Cualquiera de estas opciones es válida para poder integrar una grilla en nuestro proyecto, pero es importante tomar en cuenta que cualquiera de estas soluciones debe tener al menos uno de estos principios:

- **La grilla deberá solucionar problemas**: Esto significa que aúnque sea funcional para la estética del sitio, esta deberá enforcar sus esfuerzos en resolver un problemas como la disposición del contenido o en la pocisión de los elementos.
- **La grilla será una componente de la experiencia de usuario**: La grilla aunque no es una herramienta especifica para controlar la experiencia de usuario, tiene ciertas particularidades que permitiran al usuario controlar su propia experiencia.
- **Mientras más simple sea la grilla más efectiva será**: Un atributo importante de una grilla es su simplicidad, ya que mientras más simple sean sus columnas y más precisas sean sus medidas, mejores interfaces de usuario podremos crear.

Teniendo en cuenta esto, la opción que usaremos nosotros para implementar una grilla en el proyecto será creando una propia, a modo de conocer cómo se construyen estas usando *CSS*.

Para este caso práctico crearemos un sistema de grillas utilizando una propiedad nueva de *CSS* llamada *CSS Grid*.

Este sistema de grillas es una respuesta de *CSS* a la falta de propiedades nativas que permitieran crear grillas de manera fácil y rápida sin necesidad de usar un *frameworks CSS*. En esencia, con *CSS grid* podremos crear con unas cuantas propiedades una grilla totalmente funcional y responsiva.

Instalar Firefox Developer Edition

El único problema de esta propiedad es su juventud, ya que los navegadores aún no tienen buenas herramientas que nos permitan crear e inspeccionar este sistema, exceptuando Mozilla.

Este navegador ha puesto todas sus fichas en esta nueva tecnología creando una sección dentro del inspector de elementos que nos permitirá crear un sistema de grillas de manera rápida e intuitiva.

Para poder hacer uso de esta funcionalidad deberemos descargar la última versión de **Firefox Developer Edition** ingresando a google y luego, buscando por su nombre la página de descarga.

Cuando estemos ahí deberemos presionar en el botón *Descargar* y luego guardar el navegador en nuestro computador.

Al terminar, debemos seguir los pasos de instalación y cuando termine de instalar abriremos Firefox Developer edition para comenzar a crear nuestro sistema de grillas.

Inspector de elementos de Firefox Developer Edition

Para ingresar al inspector de elementos debemos presionar derecho y luego, escoger la opción `inspeccionar elementos`.

Al ingresar veremos a la izquierda el HTML de la página, seguido por los elementos que contiene el elemento seleccionado y al lado derecho encontraremos la sección más importante para nosotros ya que, desde aquí podremos activar y conocer los elementos que componen al sistema de grillas creado con *CSS Grid*.

CSS Grid

Ahora estudiaremos la propiedad `display:grid` de css, que nos permite crear facilmente un sistema de grilla personalizado según el diseño que debamos maquetar.

Antes de comenzar a crear el sistema de grillas que implementaremos en nuestro proyecto, es importante conocer qué es *CSS grid*.

CSS Grid es básicamente un sistema de grillas, o sea, un conjunto de líneas horizontales y verticales, con la que podremos posicionar elementos dentro de una interfaz de usuario.

Para conocerla de mejor manera crearemos una nueva página HTML, que guardaremos en el escritorio. Después de haberla guardado, lo siguiente es crear la base del HTML y agregar en `<body>` lo siguiente:

- Primero crearemos un `<div>` con clase `.container` y dentro de el pondremos 10 `<div>` con clase `.item` los que tendrán un número como contenido para poder diferenciarlos.

```
<body>
  <div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
    <div class="item">7</div>
    <div class="item">8</div>
    <div class="item">9</div>
    <div class="item">10</div>
  </div>
</body>
```

- Después, agregaremos debajo de `<title>` un etiqueta `<style>` a modo dar estilo a los elementos hijos del contenedor.
- Demos un `border: 1px solid black;` para identificar a los items, cambiemos la fuente a `monospace`, demos un tamaño a las letras de `3rem` y finalmente agreguemos un fondo de color `gold`.

```
<style media="screen">
  .item {
    border: 1px solid black;
    font-family: monospace;
    font-size: 3rem;
    background-color: gold;
  }
</style>
```

- Ahora abramos la página con Firefox. Al entrar veremos los elementos sobrepuestos unos con otros. Para poder transformar estos elementos a una grilla, primero debemos saber que *CSS Grid* se compone de dos elementos fundamentales: El contenedor de la grilla y los item de la grilla.

El contenedor es el padre de los items de una gilla con *CSS grid* ya que, en el se definen como se dispondrán las filas y columnas, asi como tambien su espacio y tamaño.

Por otra parte, los ítems son los elementos hijos del contenedor que constituyen la grilla, o sea que cada ítem representa un espacio dentro de la grilla y en conjunto componen a la grilla completa.

Resumiendo lo anterior, podemos indicar que existe un gran contenedor que controla las propiedades de *CSS Grid* y dentro de este lleva otros contenedores que se pueden traducir en **divs**, que finalmente serán los que compondran esta grilla.

Dentro de las propiedades que podremos usar con *CSS grid* se encuentran:

Propiedades de contenedor

Para definir la grilla debemos agregar al contenedor la propiedad `display: grid`. Esta propiedad definirá que el tipo de *display* del contenedor será grid.

```
.container {
  display: grid;
}
```

Si recargamos y vemos en el inspector de elementos la opción `malla` , y luego visualizamos esta malla podremos ver unos números. Estos números hacen referencia a la cantidad de elementos que constituyen a la grilla.

Referencias

- Ejemplos de Grid
- Learn CSS Grid

Definir filas y columnas

Ahora para poder definir las filas y columnas de la grilla que acabamos de activar debemos utilizar las propiedades `grid-template-columns` y `grid-template-rows`. Con la propiedad `grid-template-columns` podremos definir el tamaño que queremos que tengan las columnas y con `grid-template-rows` definiremos las filas.

Por ejemplo, si queremos agregar 2 columnas a la grilla, deberemos usar la propiedad `grid-template-columns` en el contenedor y como valor definir el tamaño que tendrá, el cual podría ser de `200px` por cada una.

```
.container {
  display: grid;
  grid-template-columns: 200px 200px;
}
```

Si recargamos el navegador veremos que los elementos ahora se posicionaron en dos columnas del mismo tamaño. Si cambiamos a `300px` en el valor de la segunda columna podremos que está aumenta el tamaño.

En el caso que queramos definir el tamaño de las filas, podremos hacerlo usando `grid-template-rows`, seguido de los tamaños específicos que queramos definir.

Unidad de medida fr

Así mismo, dentro de estas propiedades no solamente podremos usar unidades absolutas como los pixeles, sino que también podremos definir otras unidades como `%` o `fr`.

`fr` es una unidad de medida específica de *CSS grid* que nos ayudará a crear grillas fluidas. Este material nos permitirá definir la fracción de espacio que queremos usar dentro de la grilla.

Por ejemplo, si quisieramos darle el mismo tamaño a las columnas que creamos podemos usar `1fr` para cada columna.

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

Esto significa que el tamaño en total de la grilla se divide en dos. Si recargamos la página podremos ver que las columnas tienen el mismo tamaño uno con la otra.

Repitiendo tamaños

Ahora, existe un valor con el cual no es necesario definir uno por uno el tamaño de las filas o columnas, ya que podremos definir el tamaño de secciones de mismas proporciones usando `repeat()`.

La notación `repeat()` nos permitirá repetir todas, varias o solamente una sección de nuestras filas o columnas.

Veamos como funciona esta notación cambiando las valores que teniamos a `repeat()` y dentro escribiremos la cantidad de veces que queremos que se repita un tamaño, que en este podría ser `3` veces y que el tamaño sea de `1fr`.

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

Si recargamos veremos que ahora tenemos `3` columnas del mismo tamaño.

Espacio entre los ítems

Por último, conoceremos una última propiedad de contenedor que nos permitirá definir el espacio que hay entre cada uno de los componentes de la grilla usando la propiedad `grid-gap`.

Agreguemos en la clase `.container` un `grid-gap` con un tamaño fijo de `20px`.

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 20px;
}
```

Si recargamos la página veremos que en la grilla aparecieron unas líneas que segmentan a las filas y columnas. Estas tienen un tamaño de `20px`.

Si queremos definir de un espacio diferente para filas y columnas lo podremos hacer agregando en `grid-gap` un primer valor con el tamaño de espacio para las filas y luego, uno para las columnas.

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 30px 20px;
}
```

Propiedades de ítems

Bien, *CSS Grid* también nos permitirá definir en donde estará posicionados los ítems dentro de la grilla.

Como hemos visto hasta el momento cuando creamos una grilla definimos los espacio o *tracks* de las grillas, pero no las líneas que tiene.

Este último punto es importante ya que, cuando trabajamos con ítems es nesesario conocer cuanto espacio vertical tiene nuestra grilla a modo de conocer en que lugar se posicionarán nuestros elementos

Para saberlo deberemos contar desde el principio de la grilla hasta el final cuántas líneas tenemos. En el caso de la grilla creada hasta el momento, esta tiene un total de 4 líneas de columnas y 3 de filas.

Ubicación de ítems

El tener claro este punto nos permitirá trabajar de mejor manera con las propiedades de items.

Si queremos definir la ubicación de un item dentro la grilla, podremos hacerlo con `grid-column-start` o `grid-row-start` si se quiere asignar desde el comienzo y `grid-column-end` o `grid-rows-end` si es desde el final de la línea.

Para conocer de mejor manera estas propiedades, agregaremos una clase que nos permita posicionar los elementos. Agreguemos esta clase hasta el tercer item.

```
<body>
  <div class="container">
    <div class="item item-1">1</div>
    <div class="item item-2">2</div>
    <div class="item item-3">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
    <div class="item">7</div>
    <div class="item">8</div>
    <div class="item">9</div>
    <div class="item">10</div>
  </div>
</body>
```

- El primer item lo posicionaremos entre la línea 1 y 4.

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

- El segundo, entre la línea 2 y 3.

```
.item-2 {
  grid-column-start: 2;
  grid-column-end: 3;
}
```

- Y finalmente, el tercero entre la línea 2 y 4.

```
.item-3 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Si recargamos veremos que los items se posicionaron en base a las líneas que tiene la grilla. El primer item tiene un tamaño que va desde la línea 1 a la 4, el segundo entre la líne 2 y 3, y el tercero de 1 a 3.

La versión corta de estas propiedades son `grid-colum` y `grid-row`.

Ahora bien, si lo que queremos es posicionar nuestros elementos abarcando un número total de columnas, tal como lo hace *Bootstrap* con su clase `.col`, podremos usar un valor llamado `span`.

Cambiemos el valor de los tres elementos del ejercicio anterior por un `grid-column` de `span 3`, `span 2` y `span 1`, como se muestra a continuación.

```
.item-1 {
  grid-column: span 4;
}

.item-2 {
  grid-column: span 3;
}

.item-3 {
  grid-column: span 2;
}
```

Si recargamos veremos que los elementos se posicionaron basados en las columnas que definimos como valor 4, 3 o 2, lo que es muy útil si queremos que elementos dentro de nuestro *layout* se distribuyan en base a la columna, tal como lo hace *Bootstrap* con su sistema de grillas.

En conclusión, trabajar con *CSS grid* nos permitirá crear un sistema de grillas con muy pocas líneas de código, y en definitiva cumplir con los principios planteados para un buena opción de grillas.

Por otra parte, lo visto en esta sección es una pincelada de todo lo que podremos hacer con *CSS grid*. Sin embargo, con todas las propiedades que conocimos podremos construir de manera muy sencilla una grilla para nuestro proyecto.

Si quieres aprender más sobre esta tecnología te recomiendo que revises las lecturas complementarias incluidas en esta unidad a modo de conocer en mayor profundidad las propiedades y usos que nos permite hacer *CSS grid*.

Lectura complementaria

- Chris House (CSS Tricks) - A complete Guide to Grid
- Jen Simmons - Experimental Layout with CSS grid

Creando una grilla con *CSS Grid*

En esta oportunidad aprenderemos a crear nuestro propio sistema de grillas basado en el framework css *Bootstrap*, la grilla que creemos en esta parte será la grilla que se usará en los próximos desafios.

Luego de conocer algunos las propiedades básicas que nos permitirán crear nuestra grilla, es momento de implementar esta dentro del proyecto.

Para ello crearemos un sistema de grillas similar al de *Bootstrap* a modo de familiarizarnos rápidamente con la nomenclatura y uso de esta.

Para hacerlo, primero crearemos un nuevo parcial dentro dentro del directorio `layout` llamado `_grid.scss`.

Después iremos a manifiesto a modo de agregar este parcial en la hoja de estilo.

```
// Layout
@import 'layout/grid';
@import 'layout/header';
@import 'layout/main';
@import 'layout/footer';
```

Dentro de este agregaremos tres componentes específicos que contiene la grilla de *Bootstrap* que son los contenedores, *rows* y *cols*.

Creando el *container*

Comencemos creando la clase `.container` la cual contendrá dentro un `margin` derecho e izquierdo automático y un `padding` derecho e izquierdo de `15px`.

```
 /* Container
  ========================================================================
*/
.container {
  margin-left: auto;
  margin-right: auto;
  padding-left: 15px;
  padding-right: 15px;
}
```

Luego, definiremos un tamaño específico para los contenedores. Comenzaremos con un *media query* para tamaños medianos de un ancho mínimo de `768px`. En este tamaño el contenedor tendrá un ancho de `750px`.

```
@media (min-width: 768px) {
  .container {
    width: 750px;
  }
}
```

Haremos lo mismo con los tamaños grandes (`992px`) con un *container* que tendrá un ancho de `970px` .

```
@media (min-width: 992px) {
  .container {
    width: 970px;
  }
}
```

Y finalmente, para tamaños extra grandes (`1200px`) le daremos un ancho de `1170px` .

```
@media (min-width: 1200px) {
  .container {
    width: 1170px;
  }
}
```

Creando los *rows*

Ahora seguiremos creando los estilos de la clase `.row` , que será el contenedor principal de la grilla. Agreguemos dentro de esta clase un `display: grid` y luego, definimos la cantidad de columnas que tendrá nuestra grilla, que si seguimos la cantidad de columnas que tiene *Bootstrap*, serán 12. Para ello usaremos un `grid-template-columns: repeat(12, 1fr);` .

```
//* Row
/*============================================================================
*/
.row {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
}
```

Creando los *cols*

El último elemento que nos falta por definir en nuestra grilla serán la clase `.col` , que nos permitirá distribuir los elementos a lo largo de la página.

Para hacerlo debemos tener en cuenta que *Bootstrap* tiene 4 clases que afectan a los elementos dependiendo del tamaño del dispositivo.

Las clases que podremos encontrar son: `col-xs` , `col-sm` , `col-md` y `col-lg` .

Comencemos creando las clases `col-xs` .

- Agreguemos la clase `.col-xs-12` y dentro agregaremos un `grid-column` de `span 12` .

```
.col-xs-12 {
  grid-column: span 12
}
```

- Haremos lo mismo para los otros 11 `.col` que nos quedan por agregar.

```
//* Columns
/*========================================================================
*/
.col-xs-12 {
  grid-column: span 12
}

.col-xs-11 {
  grid-column: span 11;
}

.col-xs-10 {
  grid-column: span 10
}

.col-xs-9 {
  grid-column: span 9
}

.col-xs-8 {
  grid-column: span 8
}

.col-xs-7 {
  grid-column: span 7
}

.col-xs-6 {
  grid-column: span 6
}

.col-xs-5 {
  grid-column: span 5
}

.col-xs-4 {
  grid-column: span 4
}

.col-xs-3 {
  grid-column: span 3
}

.col-xs-2 {
  grid-column: span 2
}

.col-xs-1 {
  grid-column: span 1
}
```

- Luego, con los `.col-sm` haremos lo mismo, pero agregando un *breakpoint* para dispositivos medianos de `768px`.

```
@media (min-width: 768px) {
  .col-sm-12 {
    grid-column: span 12
  }

  .col-sm-11 {
    grid-column: span 11;
  }

  .col-sm-10 {
    grid-column: span 10
  }

  .col-sm-9 {
    grid-column: span 9
  }

  .col-sm-8 {
    grid-column: span 8
  }

  .col-sm-7 {
    grid-column: span 7
  }

  .col-sm-6 {
    grid-column: span 6
  }

  .col-sm-5 {
    grid-column: span 5
  }

  .col-sm-4 {
    grid-column: span 4
  }

  .col-sm-3 {
    grid-column: span 3
  }

  .col-sm-2 {
    grid-column: span 2
  }

  .col-sm-1 {
    grid-column: span 1
  }
}
```

- Haremos lo mismo con las clases `.col-md` con un tamaño `992px`.

```
@media (min-width: 992px) {
  .col-md-12 {
    grid-column: span 12
  }
```

```
  .col-md-11 {
    grid-column: span 11;
  }

  .col-md-10 {
    grid-column: span 10
  }

  .col-md-9 {
    grid-column: span 9
  }

  .col-md-8 {
    grid-column: span 8
  }

  .col-md-7 {
    grid-column: span 7
  }

  .col-md-6 {
    grid-column: span 6
  }

  .col-md-5 {
    grid-column: span 5
  }

  .col-md-4 {
    grid-column: span 4
  }

  .col-md-3 {
    grid-column: span 3
  }

  .col-md-2 {
    grid-column: span 2
  }

  .col-md-1 {
    grid-column: span 1
  }
}
```

- Y, terminaremos de crear los *cols* con la clase `col-lg` de un ancho mínimo de `1200px`.

```
@media (min-width: 1200px) {
  .col-lg-12 {
    grid-column: span 12
  }

  .col-lg-11 {
    grid-column: span 11;
  }

  .col-lg-10 {
```

```
      grid-column: span 10
    }

    .col-lg-9 {
      grid-column: span 9
    }

    .col-lg-8 {
      grid-column: span 8
    }

    .col-lg-7 {
      grid-column: span 7
    }

    .col-lg-6 {
      grid-column: span 6
    }

    .col-lg-5 {
      grid-column: span 5
    }

    .col-lg-4 {
      grid-column: span 4
    }

    .col-lg-3 {
      grid-column: span 3
    }

    .col-lg-2 {
      grid-column: span 2
    }

    .col-lg-1 {
      grid-column: span 1
    }
  }
```

Implementando grilla en proyecto

Luego de terminar de construir la grilla, es momento de recordar como aplicar la grilla de *Bootstrap* en un proyecto.

Como sabemos la estructura típica de un contenedor con grilla es:

```
<div class="container">
  <div class="row">
    <div class="col-sm-6">
      <!-- Contenido -->
    </div>
    <div class="col-sm-6">
      <!-- Contenido -->
    </div>
  </div>
</div>
```

Para poder aplicar la grilla en nuestros elementos deberemos tomar en cuenta esta estructura.

Ejemplifiquemos aquello agregando la grilla dentro de la sección `hero-section`.

Ejemplo de grilla

Primero es importante definir que la etiqueta `<section>` con clase `hero-section` es el contenedor principal de la sección, de modo que el `.row` de la grilla deberá ir al interior de la sección.

Agreguemos un `<div>` con clase `.row` y dentro de este movamos los elementos que componen a la sección.

```
  <!-- Hero Section -->
    <section class="hero-section">
      <div class="row">
        <div class="hero-section__featured-image"></div>
        <div class="hero-section__container">
          <div class="hero-section__main-content">
            <h1>Disfruta del Sabor Artesanal</h1>
            <p>Hoy, como en el pasado, usamos recetas familiares tradicionales
que están libres de sabores artificiales, colorantes o potenciadores de sabor
químicos.</p>
            <div class="hero-section__button">
              <button type="button" class="button button_primary">Ver
más</button>
            </div>
          </div>
          <div class="hero-section__info_small">
            <a href="#">Nuevos Sabores</a>
            <p>Revisa los nuevos sabores que tenemos para esta temporada.</p>
          </div>
          <div class="hero-section__info_medium">
            <a href="#">Recetas y Postres</a>
            <p>Crea las mejores recetas usando nuestro helados. Realmente te
maravillarás.</p>
          </div>
        </div>
      </div>
    </section>
```

Ahora, es momento de ir agregando los *cols* necesarios para cumplir con el diseño propuesto por nosotros.

Es importante mencionar que el hecho de reformular el diseño como lo hicimos, puede hacer que sea necesario cambiar algunos estilos a medida que vayamos agregando la grilla que creamos, de modo que para este caso práctico usaremos los *cols* para modelar el diseño.

Teniendo esto claro, es momento de seguir con la integración de los *cols*.

En el primer `<div>` este elemento sólo se muestra cuándo el tamaño es extra grande, de modo que la clase que deberemos usar para este caso será `col-lg-6`.

```
<div class="hero-section__featured-image col-lg-6"></div>
```

El siguiente elemento contiene el contenido principal de la sección, junto con lo bloque de información de colores.

Aquí lo que deberemos hacer es **definir el flujo de los elementos**. En dispositivos pequeños todos los elementos de esta sección tiene el `100%` del ancho del ancho del *viewport*, en tamaños medianos sólo el contenido principal tiene un ancho del `100%` , mientras que los otros elementos tienen un `40%` y un `60%`, y en tamaño grandes esta distribución se mantiene a escepción del contenedor principal que debe compartir el su espacio con la imagen principal.

Para lograr esto debemos definir el tamaño del conenedor del contenido principal, para ello primero agregaremos un `.row` para luego, posicionar de manera más fácil el contenido dentro de este contenedor.

En este mismo, agregaremos la clase `col-xs-12 col-lg-6` para que en el contenedor fluyan conforme a lo dispuesto en el diseño.

```
<div class="hero-section__container row col-xs-12 col-lg-6">
  <div class="hero-section__main-content">
    <h1>Disfruta del Sabor Artesanal</h1>
    <p>Hoy, como en el pasado, usamos recetas familiares tradicionales que
están libres de sabores artificiales, colorantes o potenciadores de sabor
químicos.</p>
    <div class="hero-section__button">
      <button type="button" class="button button_primary">Ver más</button>
    </div>
  </div>
  <div class="hero-section__info_small">
    <a href="#">Nuevos Sabores</a>
    <p>Revisa los nuevos sabores que tenemos para esta temporada.</p>
  </div>
  <div class="hero-section__info_medium">
    <a href="#">Recetas y Postres</a>
    <p>Crea las mejores recetas usando nuestro helados. Realmente te
maravillarás.</p>
  </div>
</div>
```

Ahora, al interior de este contenedor definiremos el flujo del contenido principal a 12 columnas y los bloques de información tendrán los dos dispositivos pequeños 12 columna y luego, en tamaños medianos cambiarán a 5 columnas, para el primero, y 7 para el segundo.

```
  <!-- Hero Section -->
    <section class="hero-section">
      <div class="row">
        <div class="hero-section__featured-image col-lg-6"></div>
```

```
          <div class="hero-section__container row col-xs-12 col-lg-6">
          <div class="hero-section__main-content col-xs-12">
            <h1>Disfruta del Sabor Artesanal</h1>
            <p>Hoy, como en el pasado, usamos recetas familiares tradicionales
que están libres de sabores artificiales, colorantes o potenciadores de sabor
químicos.</p>
            <div class="hero-section__button">
              <button type="button" class="button button_primary">Ver
más</button>
            </div>
          </div>
          <div class="hero-section__info_small col-xs-12 col-sm-5">
            <a href="#">Nuevos Sabores</a>
            <p>Revisa los nuevos sabores que tenemos para esta temporada.</p>
          </div>
          <div class="hero-section__info_medium col-xs-12 col-sm-7">
            <a href="#">Recetas y Postres</a>
            <p>Crea las mejores recetas usando nuestro helados. Realmente te
maravillarás.</p>
          </div>
        </div>
      </div>
    </section>
```

Finalmente, para omitir problemas referentes a la distribución de los bloques, eliminaremos el ancho que hayamos definido a estos.

Cuando terminemos de borrar estas reglas, recarguemos la página y veamos el resultado.

Imagen 2. Maqueta usando CSS grid.

Como podemos ver el resultado usando *CSS grid* es muy bueno y cumple con lo dispuesto en el diseño que propusimos hace algunas unidades atrás.

Ahora te toca a ti terminar de completar la implementación de la grilla en el proyecto a modo de comprender de mejor manera como funciona *CSS grid*.

Componentes de un diseño responsivo: Imágenes responsivas

Ahora aprenderemos a cómo dar el comportamiento o técnicas para que una imagen sea responsive en nuestros proyectos.

Anteriormente implementamos el segundo componente de un diseño responsivo, que era la grilla, ahora vamos a conocer un último elemento que compone este tipo de diseño conocido como **imágenes responsivas**.

¿Qué son las imágenes responsivas?

Las imágenes responsivas son tipos de imágenes que escalan su tamaño a modo de fluir con el espacio disponible que estas tengan.

Para lograrlo podremos usar diferentes estrategias que nos permitirán transformar simples imágenes estáticas a imágenes fluidas o responsivas.

Estrategias para transformar imágenes a responsivas

Uso de anchos y porcentajes

La primera técnica que veremos es usada frecuentemente para transformar imágenes estática a responsivas. Esta consta de agregar un ancho definido por porcentajes y un alto automático.

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

Esto hace que el tamaño de la imagen aumente o disminuya dependiendo del ancho del *viewport*. También, es común encontrar una versión alternativa de esta estrategia difiniendo el ancho como ancho máximo.

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

En nuestra maqueta esta técnica se encuentra aplicada en el parcial `_reset` afectando a todas las imágenes del sitio.

Background escalable

Otra estrategia interesante a aplicar en nuestros proyectos es el uso de imágenes de fondo para transformarlas en responsivas.

Para hacerlo debemos primero definir una imagen de fondo con algunas propiedades base para no repetir la imagen y para posicionar correctamente la imagen en el contenedor.

```
.hero-section__featured-image {
  display: none;
  background-image: url(../images/main-image.jpg);
  background-repeat: no-repeat;
  background-position: center;
}
```

Ahora que ya tenemos la base, debemos decidir como se dispondrá la imagen fondo. Para ello usaremos la propiedad `background-size`.

Los valores que se pueden usar para escalar o reducir el tamaño del *background* serán:

- **100% 100%**: Con estos valores podremos definir que el tamaño del fondo se estirará en un 100% del tamaño del contenido.
- ***Cover***: Esta opción permitirá escalar la imagen cubriendo todo el ancho que tenga disponible el contenedor.
- ***Contain***: Con esta opción la imagen de fondo escalará e intentará ajustarse al área del contenido manteniendo su relación de aspecto.

```
.hero-section__featured-image {
  display: none;
  background-image: url(../images/main-image.jpg);
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
}
```

En nuestro caso, la imagen principal de la maqueta usa esta técnica para posicionar la imagen.

Lectura complementaria

- Chris Coyier - Responsive images in CSS

Menú colapsable con *CSS*

Ahora crearemos un menú colapsable como el que nos entrega *Bootstrap*, pero sólo con *css*.

Luego de conocer y poner en práctica varios conceptos relacionados a los diseños responsivos, es momento de finalizar este contenido creando un componente muy común en los diseños responsivos llamado **menú colapsable**.

¿Qué son los menús colapsables?

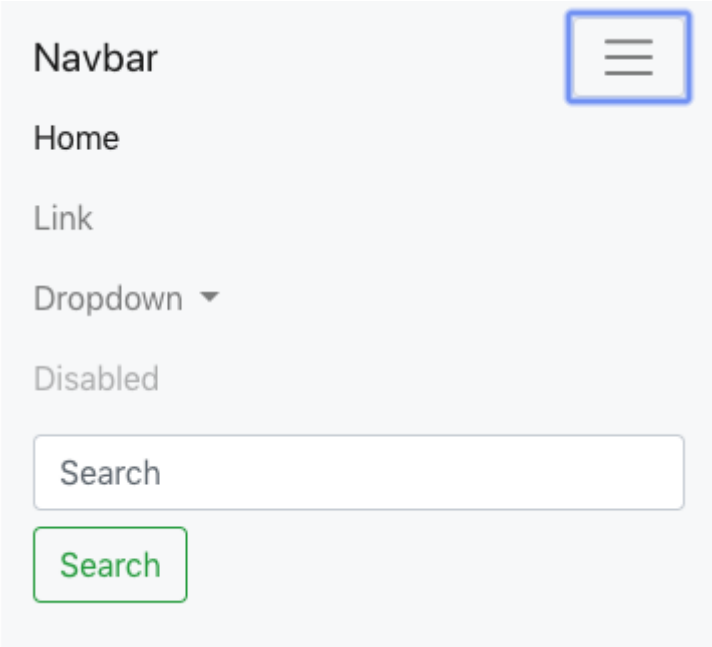

Imagen 3. Menú Colapsable.

Los menús colapsables son un tipo de navegación usada en dispositivos móviles para poder visualizar una navigación sin usar tanto espacio dentro del diseño.

Este tipo de menú es un estándar en la industria debido a que *Bootstrap* usa este tipo de navegación para tamaños responsivos.

Por lo general este tipo de menú es creado usando *JavaScript*, pero también podremos crear uno usando sólamente *CSS* y *HTML*.

Elementos de un menú colapsable

Para poder hacerlo debemos tener en cuenta que este tipo de navegación contiene algunos elementos característicos que debemos entender antes comenzar:

- **Hamburguesa**: La hamburguesa es un elemento gráfico típico de este tipo de navegación la cual se compone de tres líneas horizontales, que en conjunto forma el tan famoso elemento UI.
- ***Media queries***: Otro elemento importante es el uso de *media queries*, debido a que con ellos podremos controlar cuándo mostrar este tipo de navegación.

- ***Inputs***: Por otra parte, los *inputs* también son un elemento importante para este menú, ya que con ellos podremos obtener el evento que gatille que baje la navegación.

Bien, teniendo claro los elemento más importantes que tiene un menú colapsable, es momento de transformar nuestro menú por uno colapsable.

Reorganizando el menú de navegación

Lo primero que haremos será reorganizar la estructura que tiene nuestra navegación a modo incluir elementos importantes que componen a este tipo de menú.

Vamos a `index.hmtl` a la sección `navigation` y desde acá comenzaremos a agregar elementos y clases que nos valdrán para esta navegación. Además, es importante para trabajar de manera sencilla con la construcción el uso de una grilla.

Primero agregaremos al lado de la clase `navigation` un `.row` a modo de activar la grilla en el componente completo.

Luego, debajo de contenedor del logo crearemos un nuevo bloque el cual contendrá los elementos que mencionamos hace un momento como el *input* y la hamburguesa. Este bloque tendrá una clase `.collapsible`, además este tendrá una clase `col-xs-12` para definir que ese contenedor ocupará 12 columnas.

Debajo agregaremos un *input* para provocar que cuando el usuario presione la hamburguesa se muestre la navegación. Este será del tipo `type="checkbox"` y tendrá un `id="menu"`.

Seguiremos agregando una etiqueta `<label>` que tendrá un atributo `for="menu"`. Este tipo de atributo es usado en la etiqueta `<label>` para referirse a que el id esta asociado a un elemento. Esto significa que si presionamos esta etiqueta se marcará la caja de selección ya que se encontrarán asociadas con el `id="menu"`.

Dentro de este agregaremos la hamburguesa para que al presionar se active el menú. Para tal motivo usaremos un icono de *Font Awesome*, de modo que ingresaremos a su página y presionaremos en `icons` y finalmente buscaremos la hamburguesa escribiendo la palabra clave `hamburguer`. Escogamos el icono `bars` y copiemos su código código HTML. Finalmente, peguemos el icono dentro de la etiqueta `<label>`.

```
<!-- Header -->
<header class="header">
  <!-- Navigation -->
  <nav class="navigation row">
    <div class="navigation__logo">
      <img src="assets/images/logo.png" alt="Logo heladerías Doñihue">
    </div>
    <div class="collapsible col-xs-12">
      <input type="checkbox" id="menu">
      <label for="menu"><i class="fas fa-bars"></i></label>
      <div class="navigation__container">
        <ul class="navigation__list">
          <li class="navigation__item"><a href="#">Nosotros</a></li>
          <li class="navigation__item"><a href="#">Heladerías</a></li>
          <li class="navigation__item"><a href="#">Noticias</a></li>
          <li class="navigation__item"><a href="#">Contacto</a></li>
        </ul>
      </div>
```

```
    </div>
  </nav>
</header>
```

Agregando estilos a la barra de navegación

Ahora que se encuentran todos los elementos y clases necesarias pare el menú comenzaremos a crear los estilos que definirán a la barra de navegación.

Lo primero que haremos será definir los estilo para dispositivos móviles a modo de seguir el método *mobile first*.

Estilos dispositivos móviles

Primero demos los estilos a la clase `.navigation`. Dentro esta clase agregaremos un `position: relative`, a modo de posicionar el elemento `.navigation__logo` de manera absoluta.

```
.navigation {
  position: relative;
}
```

Luego de esto, agregaremos la clase `.navigation__logo` un `position: absolute;` para mover el logo dentro del contenedor. Este tendrá un `top: .9em` y un `left: 2.5em`.

```
.navigation__logo {
  position: absolute;
  top: .9em;
  left: 2.5em;
}
```

Seguiremos con los estilos del contenedor `.collapsible`. Dentro de este bloque se encuentran todos los elemento que creamos como la hamburguesa y la navegación misma.

Comencemos por alinear los elementos de este contenedor a la derecha usando un `text-align: right`. Debajo haremos un *nesting* agregando el elemento `label`, en el cual pondremos un `display: block` para cambiar sus propiedades, luego definiremos un `padding-top: .5em`, un `padding-bottom: .5em;`, un `padding-right: 1.5em;` y finalmente un tamaño de fuente de `1.8rem`.

```
.collapsible {
  text-align: right;

  label {
    display: block;
    padding-top: .5em;
    padding-bottom: .5em;
    padding-right: 1.5em;
    font-size: 1.8rem;
  }
}
```

Debajo agregaremos el *id* del *input* (`menu`) a modo de esconder esto de la página usando un `display:none` .

```
#menu {
  display: none;
}
```

Ahora crearemos el efecto de hacer bajar la navegación cada vez que el menú se encuentre activado usando dos elementos: Primero agregaremos un `input` que contendrá el *pseudo selector* `:checked` que nos permitirá agregar una acción cuando el `checkbox` se encuentre activado. Depués agregaremos el selector hermano `~` a la clase `.navigation-container` .

Este selector nos permite seleccionar un elemento hermano para que se apliquen los mismos estilos que el. En este caso como `input` se encuentra en el mismo elemento padre que `.navigation-container` los dos se verán afectados. Dentro de ellos agreguemos un `max-height: 100%` para que la navegación se vea cuando se presione el input.

```
input:checked ~ .navigation__container {
  max-height: 100%;
}
```

Bien, sigamos agregando estilos a `.navigation__container` , definiendo un alto de `0` al contenedor a modo de crear el efecto de colapso de la navegación. Además agregaremos una propiedad que nos permitirá prevenir el desbordamiento del contenido llamada `overflow` , la cual tendrá un valor `hidden` para evitar que se muestre este contenido fuera de la caja.

```
.navigation__container {
  max-height: 0;
  padding: 0;
  overflow: hidden;
}
```

Quitemos el *bullet* que tienen los elementos de la lista (`.navigation-list`) usando la propiedad `list-style-type:none` .

Ahora, agreguemos los estilos al enlace de la navegación agregando la clase `.navigation__item` y dentro de este el elemento `a` . Primero cambiemos el comportamiento de `a` con `display: block` , luego alineemos a la izquierda el contenido, agreguemos un `padding: .5em` , además una familia de fuentes `$open-sans` , con un tamaño de `1.2rem` , un peso de `$bold` , un color `$black` y quitemos la línea por debajo del enlace con `text-decoration: none` .

Lo que nos falta es darle un `:hover` al enlace usando `&:hover` y definiremos el color del *hover* como `$nickel` .

```
.navigation__item {
  a {
    display: block;
    text-align: left;
    padding: .5em;
    font-family: $open-sans;
    font-size: 1.2rem;
    font-weight: $bold;
    color: $black;
    text-decoration: none;
```

```
    &:hover {
      color: $nickel;
    }
  }
}
```

Si recargamos, veremos que como resultado nos quedo una menú colapsable totalmente funcional.

Estilos para escritorio

Ahora que nuestra barra de navegación es funcional en móviles, deberemos hacer lo mismo, pero esta ves en dispositivos de escritorio.

```
@media only screen and (min-width: 992px) {...}
```

Comencemos agregando un `@media` con un tamaño mínimo de `992px`.

Primero, sobre escribamos la posición de `navigatio__logo` a `top: 1em` y `left: 3em`

```
.navigation__logo {
  top: 1em;
  left: 3em;
}
```

Luego, daremos esconderemos la hamburguesa de este tamaño agregando al `label` dentro de la clase `.collapsible`, la regla `display:none`.

```
.collapsible {
  label {
    display: none;
  }
}
```

Seguiremos con la clase `navigation__container`. Para este tamaño devolveremos el alto total al contenedor escondido, además le daremos un espacio al contenedor de `padding:.5em 3em .5em 0`.

Para mostrar el contenido nuevamente usaremos `overflow`, pero esta ves con un valor de `visible`.

```
.navigation__container {
  max-height: 100%;
  padding: .5em 3em .5em 0;
  overflow: visible;
}
```

Ahora agregaremos la clase `.navigation__list`. En ella usaremos una propiedad muy útil llamada `display: flex`, la que nos permitirá transformar nuestra caja normal en una caja flexible.

Si revisamos en navegador agregando esta propiedad los elementos cambiaron su posición de vertical a horizontal. Si queremos mover la navegación, podemos usar `justify-content` y definir que los elementos se justifiquen a la izquierda (`flex-end`).

Finalmente, devolvamos el tamaño base de `padding` y `margin` usando el valor `inherit`.

```
.navigation__list {
  display: flex;
  justify-content: flex-end;
  margin: inherit;
  padding: inherit;
}
```

Por último, agreguemos unos cuantos cambios a los enlaces de la clase `.navigation__item` agregando un `padding: .5em 1em;` cambiando el tamaño de la fuente a `1rem` y cambiando el peso de la fuente a `$regular`.

```
.navigation__item {
    a {
      padding: .5em 1em;
      font-size: 1rem;
      font-weight: $regular;
    }
  }
```

Al recargar y cambiar el tamaño del *viewport* veremos que nuestra barra de navegación se encuentra totalmente lista.

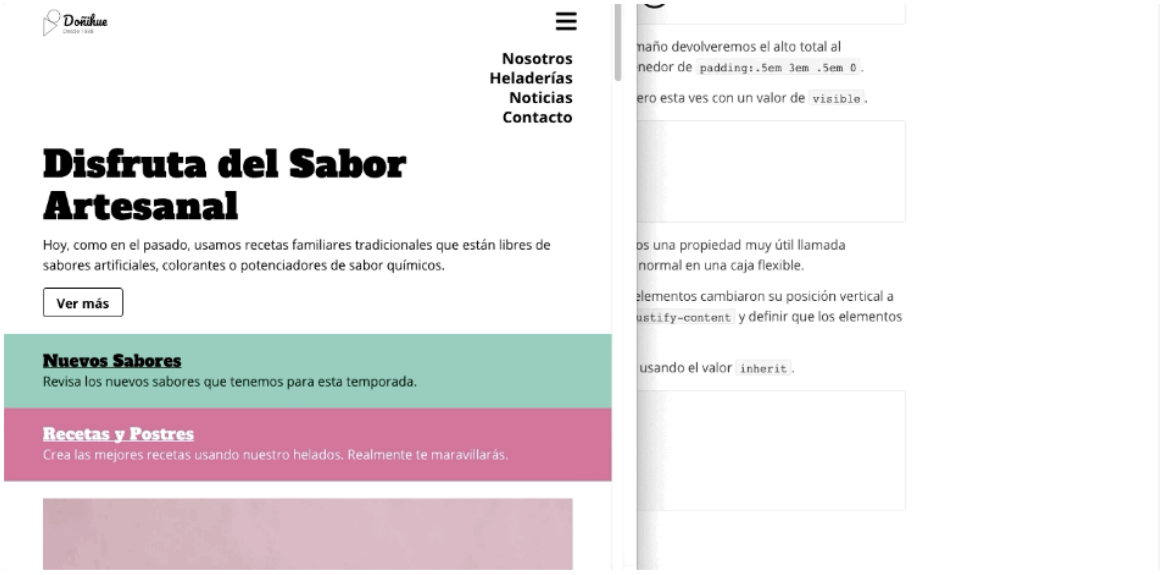

Imagen 4. Ejemplo de modificación de barra de navegación.

Retrocompatibilidad con *Modernizr*

Cuando trabajamos con propiedades y tecnologías nuevas de *CSS* como *CSS Grid* o *Flexbox* puede ocurrir que nuestra maqueta no sea compatible con algunos navegadores provocando que la experiencia de usuario sea errática y no cumpla con los objetivos comprometidos en el proyecto.

Para evitar que suceda esto en nuestros proyectos es importante tener en consideración que existe herramienta que nos ayudará a mejorar la compatibilidad que tiene nuestra maqueta con los navegadores llamada *Modernizr*.

¿Qué es *modernizr*?

Modernizr es una herramienta que detecta automáticamente las capacidades que tiene un navegador para mostrar las últimas tecnologías ofrecidas en la web.

En concreto, esta librería usa una característica llamada `feature detection` que permite conocer si el navegador tiene las capacidades para mostrar una característica especifica. En el caso que el navegador no pueda renderizar una característica, *Modernizr* agregará una clase para emular dicha característica.

¿Cómo instalar modernizr en nuestro proyecto?

Para instalar *Modernizr* en nuestro proyecto debemos ingresar a la página oficial de la librería y luego, presionar en la opción `Development Build`.

Aquí veremos varias opciones para detectar diversos elementos del navegador. En nuestro caso práctico elegiremos todas las funcionalides a modo de conocer su funcionamiento. De igual manera, es importante tener en cuenta que cuando el sitio vaya a producción deberemos seleccionar sólo funcionalidades necesarias, a modo de optimizar la performance de nuestro sitio.

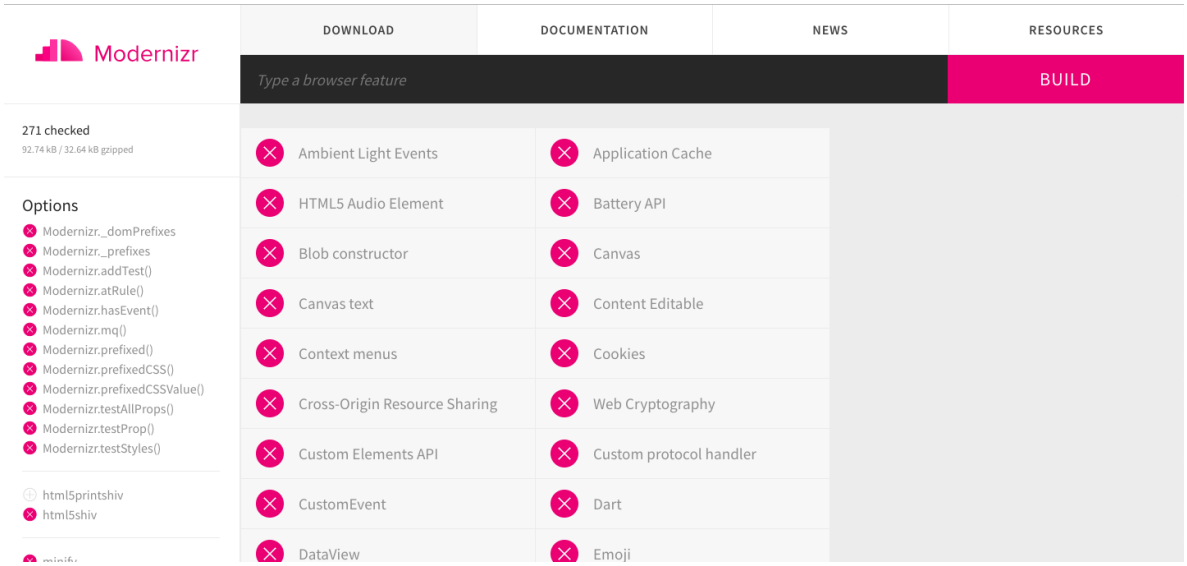

Imagen 5. Modernizr.

Luego, cuando terminemos de escoger todas las opciones debemos presionar en el botón `build` y luego, seleccionar la forma en que queremos descargar el archivo. En nuestro caso elegiremos la descarga de la opción `build`.

Cuando termine de descargar debemos crear una nuevo directorio dentro de `assets` llamado `js`. En esta moveremos el archivo descargado.

Finalmente, integraremos este archivo a nuestra maqueta usando la etiqueta `script` y agregando en ella la ruta relativa del archivo. Esta etiqueta, según explica la documentación de *Modernizr* deberá estar dentro de la etiqueta *head*, a modo de evitar comportamientos no deseados.

```
<script src="assets/js/modernizr-custom.js"></script>
```

Modernizr funcionando

Ahora que hemos integrado *Modernizr* a nuestro proyecto podremos ver de mejor manera como actua esta librería.

Cuando ingresemos a nuestro navegador y recarguemos la página veremos en el inspector de elementos varias clases en la etiqueta `<html>` que antes no se encontraban.

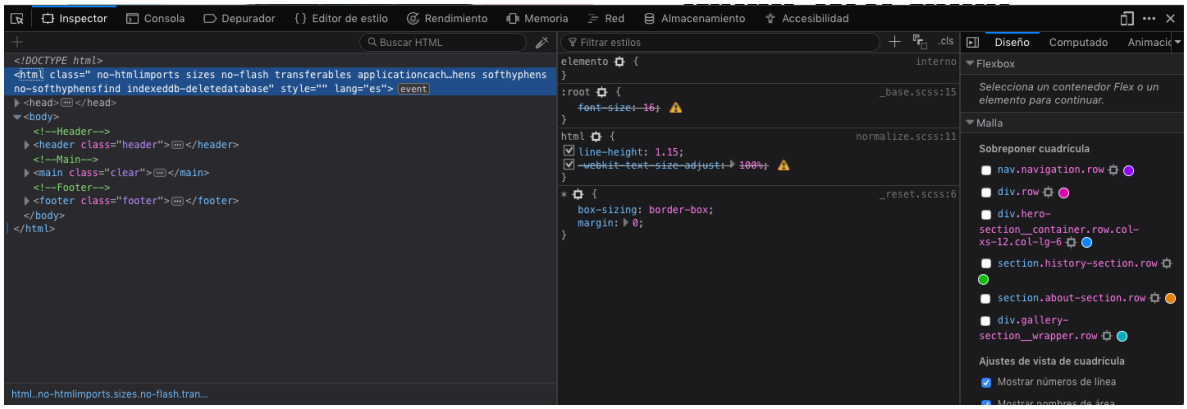

Imagen 6. Visualización del inspector de elementos.

Estas clases son creadas por *Modernizr* cuando detecta una funcionalidad de *HTML5* o *CSS3* con las que el navegador es incompatible. Esto significa que cuando estas clases aparecen, *Modernizr* modifica el estilo por otro. A su vez si detecta que no existe una funcionalidad agregará otra clase que comienza con el prefijo `no-`.

Modernizr tiene muchísimas más soluciones, pero con lo visto es más que suficiente para nuesto proyecto.

Utilizar esta librería es una buena idea para que nuestros proyectos sean retrocompatibles a modo de homogeneizar la experiencia en los navegadores.