educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 15

Grillas

Grillas en Bootstrap

Al momento de trabajar con grillas es importante recordar que si bien se han trabajado con variantes, **en la actualidad el sistema más utilizado es el de 12 columnas.**

Flex

En este sentido, Bootstrap utiliza el sistema mencionado anteriormente, y trabaja a su vez a partir de la versión 4, con **`flex`**. Recordemos que es un **valor de display**. Su trabajo viene acompañado con dos propiedades (entre otras tantas) que son complementarias y permiten **alinear el contenido**.

Veamos un ejemplo en la próxima pantalla.

En este ejemplo, **flex** permite encolumnar elementos; **justify-content** trabaja con la alineación horizontal y mientras, **align-items** trabaja con la alineación vertical:

```
#contenedor { display: flex; justify-content: center; align-items: center;}
```

Sin embargo, podemos también invertir la dirección del **flex**. De esta manera, entonces podemos apilar los elementos cambiando la dirección con la propiedad llamada **flex-direction**:

```
#contenedor { flex-direction: column; display:flex; justify-content: center; align-items:
center;}
```

Clases utilizadas en la grilla

En Bootstrap, entender estos conceptos es fundamental para poder avanzar. Básicamente porque este framework trabaja con las mismas propiedades, pero a través de **clases**.

El sistema de 12 columnas debe entender como parte de los *breakpoints,* que son los mismos que utilizamos en el maquetado responsive, más allá de usar o no Bootstrap.

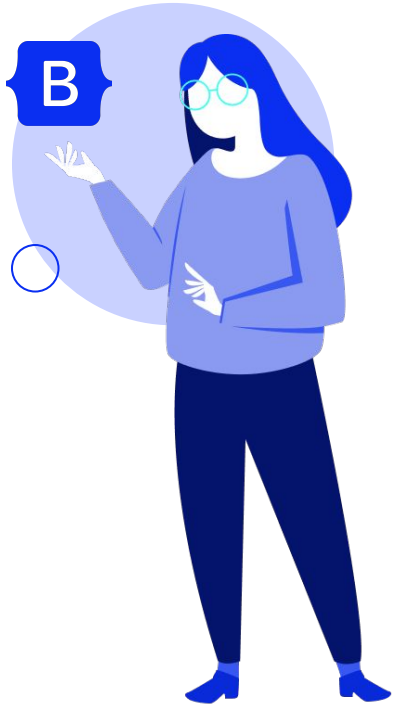
Puntualmente, para entender las clases usadas en nuestra grilla, veamos la imagen de la derecha.

Debemos siempre primero entender el **sistema de grilla en nuestro propio diseño**, por ejemplo:

En general, como ya hemos mencionado, es importante **entender este sistema de grillas** en todos los diseños (breakpoints) que hayamos realizado. A partir de eso, debemos también **aprender a utilizar las clases** correspondientes.

Las clases que detallaremos en las pantallas siguientes son:

- `col`.
- `row`.



Clase col

Esta clase nos **permite trabajar de manera automática con el ancho disponible**. Es decir, toma el espacio que haya o que quede.

En el ejemplo de la derecha, cada **`.col`** toma cuatro columnas, dado que el espacio total está solo dividido en tres partes:

```
<div class="container">
    <div class="row">
      <div class="col">
        Columna que ocupa 4
      </div>
      <div class="col">
        Columna que ocupa 4
      </div>
      <div class="col">
        Columna que ocupa 4
      </div>
    </div>
  </div>
```

Clase row

Vemos también la importancia de **`.row` que permite utilizar una fila**, que en sí contiene la propiedad **`display`** con el valor **`flex`**, para poder encolumnar los elementos anidados.

La clase `.col` (entre otras derivadas que más adelante veremos) **permite indicar en cuántas columnas se dividirá la fila.**

En el ejemplo de la derecha, hemos elegido que específicamente una columna tome el ancho de cinco, y el resto se adapte automáticamente al espacio restante.

```
<div class="container">
    <div class="row">
      <div class="col-5">
        Columna que ocupa 5
      </div>
      <div class="col">
        Columna que ocupa el espacio restante
      </div>
      <div class="col">
        Columna que ocupa el espacio restante
      </div>
    </div>
  </div>
```

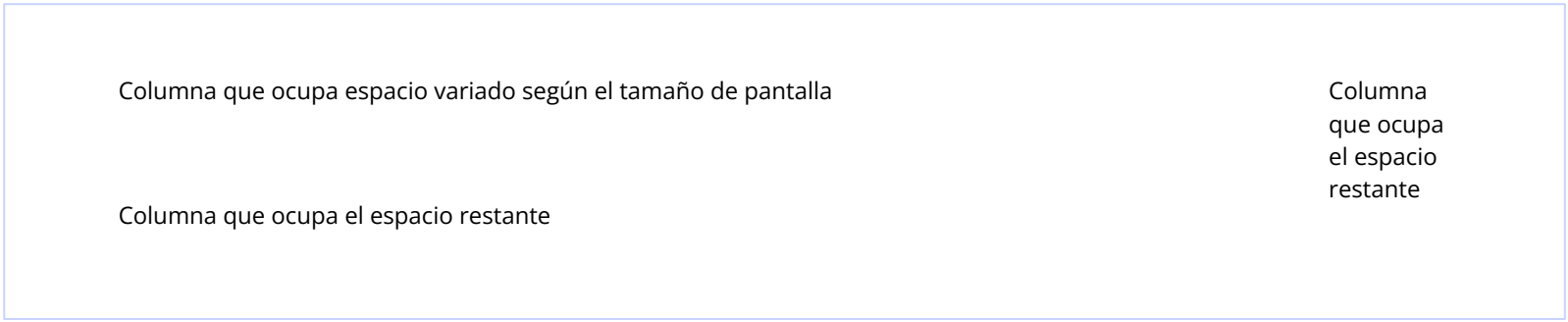
Cada breakpoint tiene su clase específica

En los diferentes diseños de nuestro prototipo, encontramos que podemos utilizarlos para **marcar cortes diferentes**, donde en un tamaño la misma columna, ocupa más espacio que en un tamaño mayor o menor. De esta forma indicamos **en cada contexto, cuánto será el espacio tomado por cada columna**.

En el ejemplo de la pantalla siguiente, veremos cómo efectivamente la misma columna varía, según el tamaño del dispositivo de origen.

```
<div class="container">
    <div class="row">
      <div class="col-6 col-sm-10 col-md-2 col-lg-11">
        Columna que ocupa espacio variado según el tamaño de pantalla
      </div>
      <div class="col">
        Columna que ocupa el espacio restante
      </div>
      <div class="col">
        Columna que ocupa el espacio restante
      </div>
    </div>
  </div>
```

Cada breakpoint indicado toma todo tamaño mayor a ese breakpoint, salvo que se indique lo contrario. En el ejemplo anterior, todo tamaño mayor que el breakpoint **lg**, generará que la primera columna se extienda, o tome 11 columnas dentro de las 12. Por esa razón, el resultado es que **al no haber espacio, la columna restante caerá a una segunda fila**:



Distribución vertical en grillas

Podemos trabajar con las **mismas propiedades que tenemos en css de `flex`**, desde Bootstrap.

Por ejemplo, para alinear verticalmente un contenido, se trabaja como vemos en la imagen de la derecha. Se puede trabajar con valores similares a los conocidos.

```
<div class="row align-items-start">
    <div class="col">
        <h2>Tecnología para todos</h2>
    </div> </div>
```

```
<div class="row align-items-end">
    <div class="col">
        <h2>Tecnología para todos</h2>
    </div> </div>
```

Align-items

Más allá del valor de la fila o del contenedor general, podemos hacer que **una columna específica tenga otro valor**, al igual que sucede cuando trabajamos de forma directa desde nuestra hoja de estilo.

En este caso, la clase será **`align-self...`** y el valor que le corresponda, como vemos en la imagen siguiente:

```
    <header>
<div class="row  align-items-end" style="height: 400px;">
    <div  class="col align-self-center">
        <h2>Tecnología para todos</h2>
    </div> </div>
```

Distribución horizontal en grillas

Podemos trabajar con las **mismas propiedades que tenemos en css de `flex`**, desde Bootstrap.

Por ejemplo, para alinear horizontalmente un contenido, se trabaja como vemos en la imagen de la derecha. Se puede trabajar con valores similares a los conocidos.

```
<div class="row justify-content-center">
    <div  class="col">
        <h2>Tecnología para todos</h2>
    </div> </div>
```

```
<div class="row justify-content-around">
    <div  class="col">
        <h2>Tecnología para todos</h2>
    </div> </div>
```

Justify-content

Podemos trabajar como hacemos con **`margin:auto`** en las reglas, desde nuestra hoja de estilo, implementar una clase similar para manejar individualmente un elemento.

Por ejemplo, orientándose a un lado específico. En la imagen, los elementos están orientados hacia el centro, pero el último se orienta hacia la derecha (end) porque tiene **`ms-auto`**:

```
<div class="row justify-content-center ">

    <div  class="col-md-2" style="background-color: blue;">
        <h2>1</h2>
      </div>

      <div  class="col-md-4  ms-auto" style="background-color: red;">
          <h2>3</h2>
        </div>

                                    </div>
```

Gutters

También se puede trabajar con ***gutters*** **o espacios**. Es importante aclarar que esto muchas veces, tiene como consecuencia que **los elementos "se caen"** o sobrepasan a sus contenedores. Por eso, en el ejemplo de la próxima pantalla, vemos cómo se utiliza **px** (representa **ps** y **pe** conjuntamente) para poder de esa manera, resolver ese inconveniente.

Ejemplo (código y resultado):

```
<div class="container px-4 text-center">
    <div class="row gx-5">
      <div class="col">
       <div class="p-3 border bg-dark text-white">Contenido</div>
      </div>
      <div class="col">
        <div class="p-3 border bg-primary text-dark">Contenido</div>
      </div>
    </div>
  </div>
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!