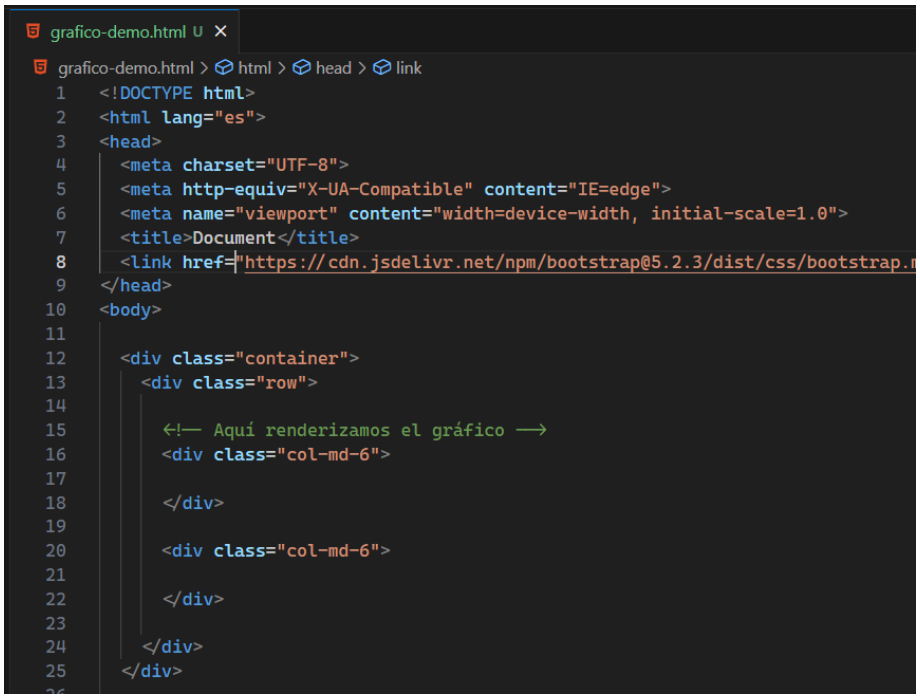
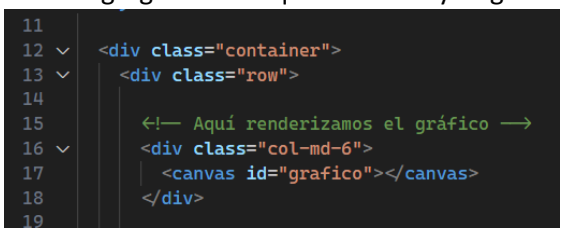
**APRENDIENDO CHART JS**

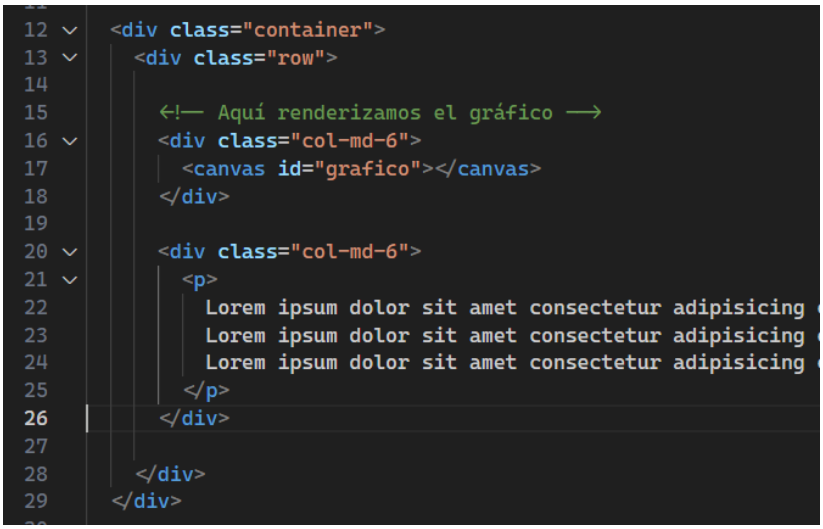
1. Diseñe un nuevo archivo HTML (**grafico-demo.html**), para este ejemplo, cree una fila y divida en dos columnas iguales (Bootstrap 5.x)



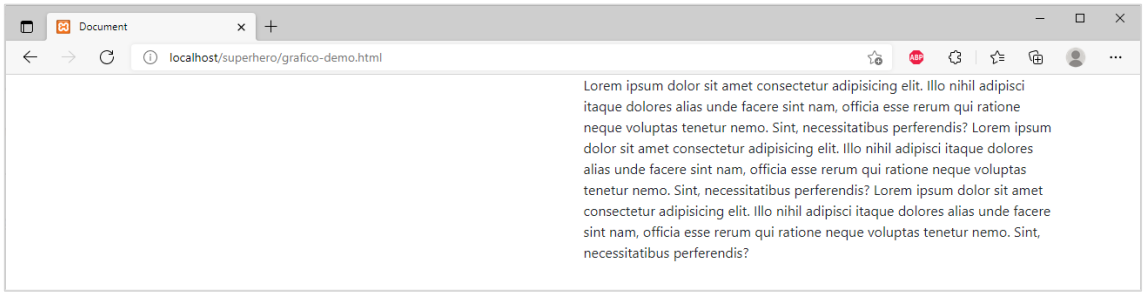

2. En la primera columna agregue una etiqueta **canvas** y asígnele un ID




3. La tercera columna es solo un espacio que puede ser utilizado para cualquier otro propósito, así que construya un párrafo con muchas líneas de prueba:



Hasta el momento debería tener



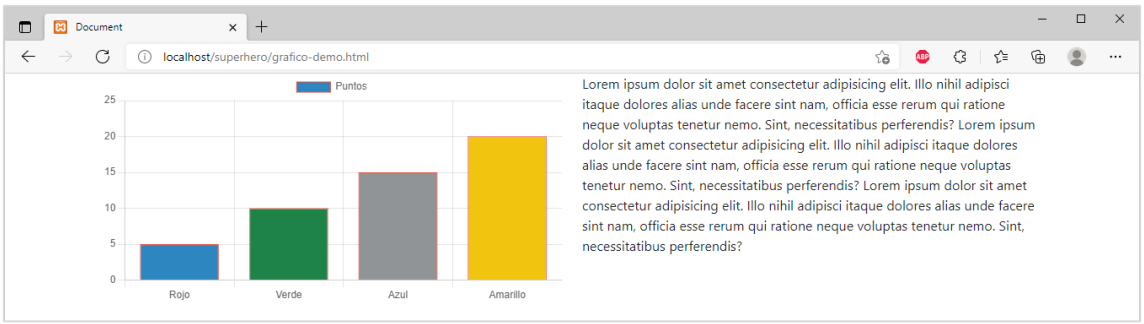

4. Incluya un bloque JavaScript, haga uso del CDN e ingrese las siguientes instrucciones:

   **CDN:**
   https://cdn.jsdelivr.net/npm/chart.js

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>
  document.addEventListener("DOMContentLoaded", () => {

    //Objeto para referencias el canvas
    const lienzo = document.getElementById("grafico");

    const graficoBarras = new Chart(lienzo, {
      type: 'bar',
      data: {
        labels: ['Rojo', 'Verde', 'Azul', 'Amarillo'],
        datasets: [
          {
            borderColor: '#E74C3C',
            backgroundColor: ['#2E86C1', '#1D8348', '#909497', '#F1C40F'],
            label: 'Puntos',
            data: [5, 10, 15, 20, 25],
            borderWidth: 1
          }
        ]
      }
    });
```

Los atributos **borderColor** y **backgroundColor** NO son obligatorios. Si los incluye, puede elegir la gama de colores que desee: https://htmlcolorcodes.com/es/

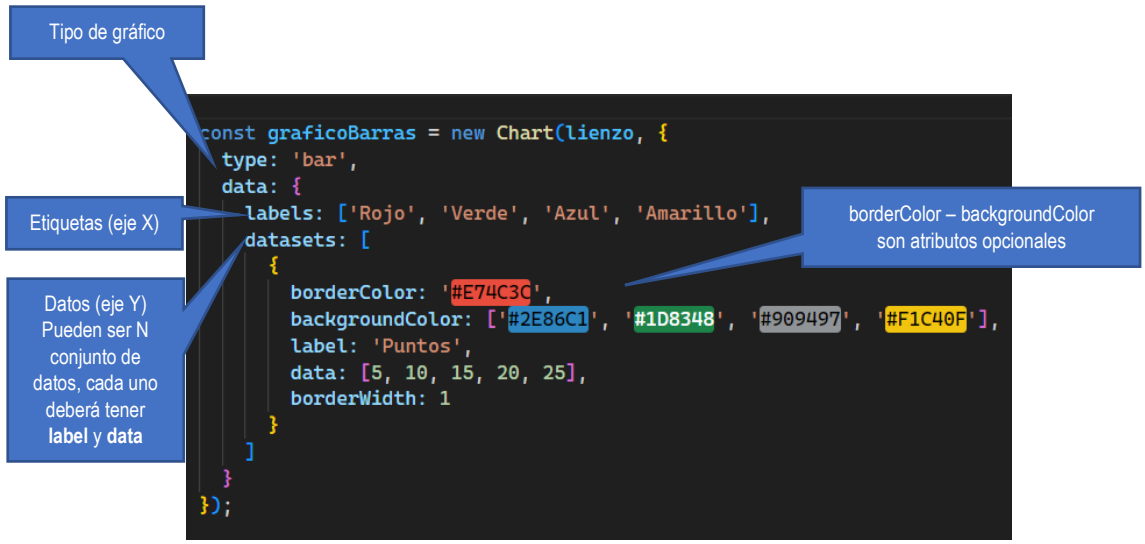
5. En el navegador



**IMPORTANTE:**

- Los atributos definidos con:
  - Atributo: ‘’ Cadena
  - Atributo: 1 Entero
  - Atributo: {} Objetos
  - Atributo: [] Colecciones
  - Atributo: [{}] Colección de objetos
  - Atributo: [‘’] Colección de cadenas

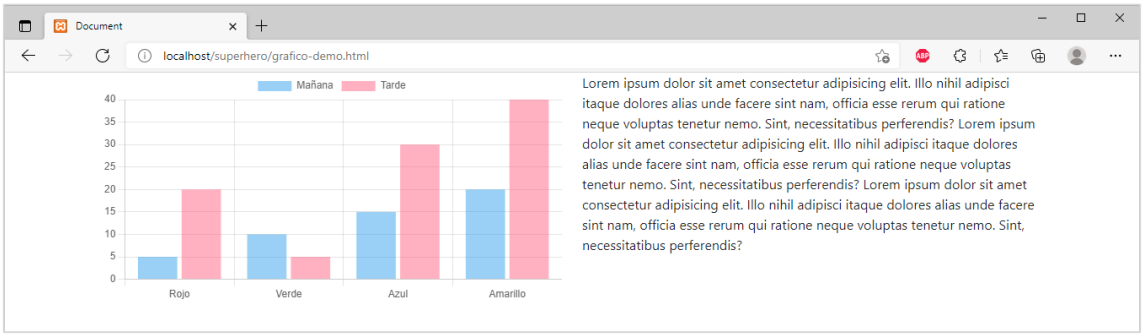
- Analizando código anterior:




6. Antes de continuar y a fin de mantener el código lo más limpio posible, eliminaré borderColor, backgroundColor y borderWidth (si Ud. Desea lo puede mantener). Para comprender mejor el concepto de datasets, agregue un nuevo set de datos:

```
33  <script>
34    document.addEventListener("DOMContentLoaded", () => {
35
36      //Objeto para referencias el canvas
37      const lienzo = document.getElementById("grafico");
38
39      const graficoBarras = new Chart(lienzo, {
40        type: 'bar',
41        data: {
42          labels: ['Rojo', 'Verde', 'Azul', 'Amarillo'],
43          datasets: [
44            {
45              label: 'Mañana',
46              data: [5, 10, 15, 20, 25]
47            },
48            {
49              label: 'Tarde',
50              data: [20, 5, 30, 40, 10]
51            }
52          ]
53        }
54      });
55
```

7. El resultado en el navegador




**ACTUALIZANDO DATOS**

Vamos a copiar el archivo anterior (grafico-demo.html) y crearemos el archivo grafico-update.html, a partir de este momento, trabajaremos con este último

1. Empezaremos agregando un formulario y 4 cajas de texto, además de un botón actualizar, estos elementos se ubicarán donde anteriormente estaba el párrafo con el texto de prueba:

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

    <!— Aquí renderizamos el gráfico —>
    <div class="col-md-6">
      <canvas id="grafico"></canvas>
    </div>

    <!— Ahora construiremos un formulario —>
    <div class="col-md-6">
      <form action="" autocomplete="off" class="mt-2">
        <div class="form-floating mb-3">
          <input type="number" class="form-control" placeholder="Valor rojo" id="rojo">
          <label for="rojo">Valor rojo</label>
        </div>
        <div class="form-floating mb-3">
          <input type="number" class="form-control" placeholder="Valor verde" id="verde">
          <label for="verde">Valor verde</label>
        </div>
        <div class="form-floating mb-3">
          <input type="text" class="form-control" placeholder="Valor azul" id="azul">
          <label for="azul">Valor azul</label>
        </div>
        <div class="form-floating mb-3">
          <input type="text" class="form-control" placeholder="Valor amarillo" id="amarillo">
          <label for="amarillo">Valor amarillo</label>
        </div>
      </form>
      <button type="button" id="actualizar" class="btn btn-info">Actualizar</button>
    </div>

  </div> <!— Fin row —>
</div> <!— Fin container —>
```

2. El siguiente paso será crear un objeto para manejar al botón

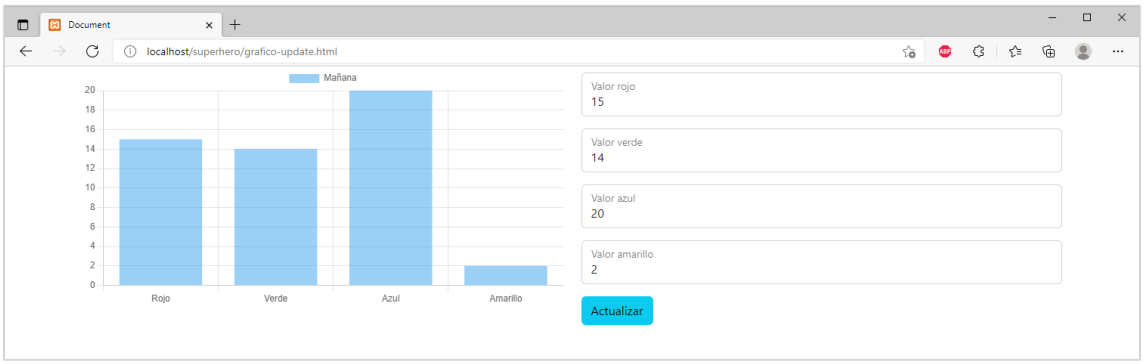
```
document.addEventListener("DOMContentLoaded", () => {

  //Objeto para referencias el canvas
  const btActualizar = document.querySelector("#actualizar");
  const lienzo = document.getElementById("grafico");
```

3. Agregue una nueva función que permita leer el valor de cada caja de texto, convirtiendo el valor a entero, estos constituirán un nuevo arreglo que reemplazará al existente.

```
function actualizarData(){
  const nuevoArreglo = [
    parseInt(document.querySelector("#rojo").value),
    parseInt(document.querySelector("#verde").value),
    parseInt(document.querySelector("#azul").value),
    parseInt(document.querySelector("#amarillo").value)
  ];

  //Asignamos el nuevo datos
  graficoBarras.data.datasets[0].data = nuevoArreglo;
  graficoBarras.update();
}

//Evento que dispara al método
btActualizar.addEventListener("click", actualizarData);
```

Fuera del método actualizarData, utilice el evento click del botón para desencadenar esta acción

4. Resultado:



**ACTIVIDAD**

Construir un gráfico estadístico, que utilice datos provenientes de una consulta. Debemos obtener el total de super héroes por bando (tabla alignment)

1. La consulta

```
55    /* Contabilizar cuántos super hérores existen por bando (alignment) */
56    DELIMITER $$
57    CREATE PROCEDURE spu_superhero_alignment_resume()
58    BEGIN
59        SELECT
60            CASE
61                WHEN  alignment.`alignment` IS NULL THEN 'Ninguno'
62                WHEN  alignment.`alignment` IS NOT NULL THEN alignment.`alignment`
63            END 'alignment',
64                COUNT(superhero.`id`) 'total'
65          FROM superhero
66          LEFT JOIN alignment ON alignment.`id` = superhero.`alignment_id`
67          GROUP BY alignment.`alignment`;
68    END $$
69
```

Donde:

- Al no tener valor de entrada, también se pudo construir como vista
- La consulta utiliza CASE WHEN, ya que algunos “superhero” no tiene un bando definido (NULL), entonces, al momento de enviar los datos esta ausencia de valor podría generar problemas.
- El primer WHEN indica que si el campo es NULL que lo cambie por “Ninguno”
- El segundo WHEN indica que si el campo NO es NULL, que lo deje como está
- Se hace un conteo (COUNT) de todos los superhero, utilizando su ID (PK)
- La consulta debe utilizar LEFT JOIN en lugar de INNER JOIN, ya que algunos registros de héroes no tienen definido su bando (NULL)
- Al utilizar la función de resumen – COUNT- es necesario indicar que el resultado debe presentarse agrupado/en grupos de bandos de (alignment)
- Para una consulta resumen, utilice solo los campos requeridos, en este caso, son solo dos campos los mostrados

Resultado obtenido

```
69
70      CALL spu_superhero_alignment_resume();
```

1 Result | 2 Messages | 3 Table Data | 4 Info

(Read Only)

| alignment | total |
|---|---|
| Ninguno | 6 |
| Bad | 212 |
| Good | 504 |
| Neutral | 28 |

2. Creamos el método en la clase contenida en SuperHero.php

```php
public function getAlignmentResume(){
  try{
    $consulta = $this->conexion->prepare("CALL spu_superhero_alignment_resume()");
    $consulta->execute();
    return $consulta->fetchAll(PDO::FETCH_ASSOC);
  }
  catch(Exception $e){
    die($e->getMessage);
  }
}
```

3. El controlador ha sido optimizado, comparto una captura completa, aunque lo único a agregar es lo señalado en la parte inferior.

superhero.php M

controller > superhero.php

You, 11 seconds ago | 1 author (You)

```php
<?php

require_once '../model/SuperHero.php';

function renderJSON($object = []){
  if ($object){
    echo json_encode($object);
  }
}      You, 11 seconds ago • Uncommitted changes

if (isset($_POST['operacion'])){

  $superhero = new SuperHero();
  if ($_POST['operacion'] == 'listar'){
    renderJSON($superhero->listarSuperHero($_POST['publisher_id']));
  }

  if ($_POST['operacion'] == 'filtrar'){
    $filtros = [
      "race_id"     => $_POST['race_id'],
      "gender_id"   => $_POST['gender_id'],
      "alignment_id"=> $_POST['alignment_id']
    ];
    renderJSON($superhero->filtrarSuperHero($filtros));
  }

  if ($_POST['operacion'] == 'resumenBandos'){
    $datos = $superhero->getAlignmentResume();
    renderJSON($datos);
  }

}
```

Se integró la función renderJSON para limpiar el código

4. Cree un nuevo archivo en la vista, lo llamaremos: “grafico-datos.html” y en el construimos lo siguiente:

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

    <!— Aquí renderizamos el gráfico —>
    <div class="col-md-7">
      <canvas id="grafico"></canvas>
    </div>

    <!— Ahora construiremos un formulario —>
    <div class="col-md-5">
      <h4>Resumen</h4>
      <ul id="lista-leyenda">
        <!— datos asíncronos —>
      </ul>
      <button class="btn btn-sm btn-success" id="actualizar">Actualizar</button>
    </div>

  </div> <!— Fin row —>
</div> <!— Fin container —>
```

5. El siguiente paso es incorporar la librería ChartJS por CDN, luego crear los objetos incluyendo al gráfico

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>
  document.addEventListener("DOMContentLoaded", () => {

    //Objeto para referencias el canvas
    const btActualizar = document.querySelector("#actualizar");
    const lienzo = document.getElementById("grafico");
    const leyenda = document.querySelector("#lista-leyenda")

    const graficoBarras = new Chart(lienzo, {
      type: 'bar',
      data: {
        labels: [],
        datasets: [
          {
            label: 'Bandos super héroes',
            data: []
          }
        ]
      }
    });
```

*Como puede observar, nuestro gráfico no tiene información asignada en **labels[]** ni en **data[]**, ambos atributos requieren colecciones/arrays*

6. Crearemos una función que permita tomar un valor de entrada (resultado de la consulta / arreglo asociativo) y renderizar el gráfico, a su vez, genera una lista con los valores a modo de leyenda

```
function renderGraphic(coleccion = []){
  let etiquetas = []; //Eje X
  let datos = [];     //Eje Y
  leyenda.innerHTML = ``;

  coleccion.forEach(element => {
    //Enviamos los datos a los arreglos
    etiquetas.push(element.alignment);
    datos.push(element.total);

    //Los renderizamos en la lista <ul>
    const tagLI = document.createElement("li");
    tagLI.innerHTML = `${element.alignment}: <strong>${element.total}</strong>`;
    leyenda.appendChild(tagLI);
  });

  //Asignamos el nuevo datos
  graficoBarras.data.labels = etiquetas;
  graficoBarras.data.datasets[0].data = datos;
  graficoBarras.update();
}
```

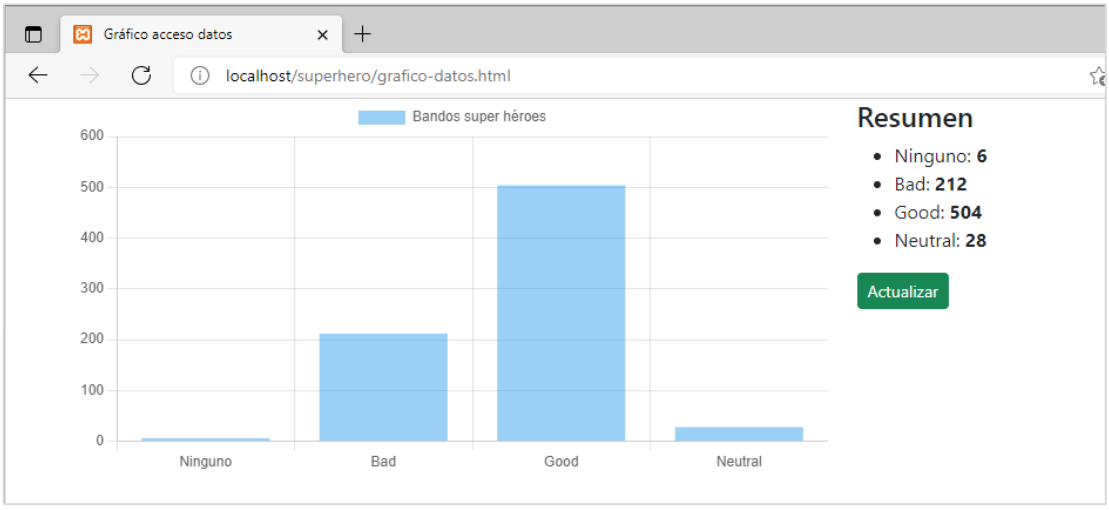
7. Finalmente, construimos una función que obtiene los datos de forma asíncrona y los envía para su renderización a la función definida en el paso 6 (renderGraphics)

```
function loadData(){
  const parametros = new URLSearchParams();
  parametros.append("operacion", "resumenBandos");

  fetch(`./controller/superhero.php`, {
    method: 'POST',
    body: parametros
  })
    .then(respuesta => respuesta.json())
    .then(datos => {
      renderGraphic(datos);
    });
}
```

Esta última función se dispara desde el botón

```
  //Evento que dispara al método
  btActualizar.addEventListener("click", loadData);

});
</script>
```

Resultado obtenido:




**TAREA:**
**GENERAR LOS SIGUIENTES GRÁFICOS:**

- Buenos y malos de una empresa publicadora seleccionada por el usuario
- Los colores de ojos se debe generar una barra para las publicadoras: MARVEL COMICS, DC COMICS
- Buenos, malos, neutrales y N/A (todos los registros de la tabla asignment) y de todas las casas publicadoras con ID: 2, 5, 11, 18 y 24
- Géneros (gender) por bando (alignment) de cada super héroe
- Total de super héroes por razas (este dato deberá seleccionarlo el usuario a través de la vista de un total de 61 opciones)

En clases se compartió los modelos referenciales a utilizar.