



ugr

Universidad
de Granada

Trabajo sobre JavaScript y ChartJS

Alberto Lejárraga Rubio

50618389N



Escuela Técnica Superior de
Ingeniería Informática y de
Telecomunicación

—
Madrid, 25 de mayo de 2021

Contenido

JavaScript	3
Código del lado del cliente frente a código del lado del servidor	3
Alternativas a JavaScript.....	3
Principales usos	4
Validación de formularios.....	4
Enviar alertas y mensajes de confirmación	5
Modificar DOM	5
Requerir información al servidor sin salir de la página (Ajax)	6
Librerías y frameworks JavaScript	6
ChartJS	7
Instalación.....	7
Configuración básica.....	7
Distintos tipos de gráfico	8
Línea.....	8
Barra	8
Radar	9
Tarta, Doughnut y Polar.....	9
Opciones avanzadas	10

JavaScript

JavaScript es un lenguaje de programación que puede ser utilizado tanto en la parte del cliente como en la parte del servidor, como se verá a continuación. Pero como en este trabajo se abordará principalmente la parte del cliente, este uso de JavaScript podría tratar de definirse como un lenguaje de programación que permite realizar acciones sobre una página web sin que intervenga necesariamente el servidor, tales como modificar el contenido de la página, modificar su estilo, reaccionar a acciones del usuario, y prácticamente cualquier cosa que se le ocurra al programador.

Código del lado del cliente frente a código del lado del servidor

JavaScript nació en 1995 para cubrir la necesidad de que el contenido de las páginas web del navegador de Netscape (predecesor de Mozilla) no fuera estático, sino que pudiera modificarse sin la necesidad de hacer una petición al servidor, es decir, que se pensó inicialmente como un lenguaje para actuar del lado del cliente.

Pero dado el gran éxito que cosechó este lenguaje, que prácticamente se ha convertido en el estándar para este propósito, como se verá en las siguientes secciones, a finales de la década de los 2000 se empiezan a realizar implementaciones de este lenguaje para llevar a cabo tareas también del lado del servidor ([aquí](#) pueden verse estas implementaciones). Entre ellas, la principal en lo que a desarrollo web se refiere es NodeJS, que es un entorno de ejecución multiplataforma.

Por lo tanto, aunque también se pueda utilizar para la parte del backend (del lado del servidor), a partir de este punto se explicará JavaScript en su uso como lenguaje del frontend (del lado del cliente).

Alternativas a JavaScript

Como se ha comentado en el apartado anterior, JavaScript fue creado en 1995, y su creador fue Brendan Eich, de la mano de la compañía Netscape. En un primer momento, dicha compañía trató de utilizar Java para poder realizar tareas de programación directamente en su navegador, pero, finalmente, decidieron hacer un lenguaje nuevo. Y dicho lenguaje es JavaScript. Java, perteneciente a Sun, por tanto, podría decirse que fue una alternativa para programar en el navegador.

En estos inicios, tuvo que competir también frente a Microsoft, que estaba tratando de integrar, del mismo modo, programación en su propio navegador, Internet Explorer. Dicha empresa realizó otra implementación de JavaScript, llamada JScript y, por otro lado, VBScript, basada en su también lenguaje Visual Basic.

De esta “guerra”, podría decirse que salió ganador JavaScript por distintos motivos, entre los cuales podrían estar que no solo funcionaba en Netscape, sino también en otros navegadores como el propio Internet Explorer, o que tenía una sintaxis más cómoda para la mayoría de programadores, pues era más similar a Java, C o C++ que a Visual Basic.

Más adelante surgirían también otras alternativas como FlashPlayer, de Adobe, aunque en la actualidad, el lenguaje más utilizado es JavaScript, al ser el lenguaje que “entienden” todos los navegadores. De hecho, hay [otros lenguajes](#) que pueden utilizarse para programar el frontend de una aplicación web, entre los que destacan Dart, TypeScript o CoffeeScript, aunque todos ellos comparten la similitud de que deben “compilarse” a JavaScript antes de ser ejecutados por el navegador, haciendo a dicho lenguaje fundamental, al menos, a día de hoy.

Principales usos

Validación de formularios

Los datos que envía un usuario a través de un formulario en una página web deben validarse para evitar inconsistencias en datos almacenados en la base de datos, o incluso, para prevenir posibles inyecciones de código que puedan afectar al servidor. Y dicha validación debe hacerse siempre en el servidor, antes de almacenarse la información en la base de datos, por si se hubiera accedido al formulario por alguna otra vía (por ejemplo, una petición GET o POST desde una línea de comandos). Pero es muy común que se realice también en la propia página web, de manera que no haya que sobrecargar al servidor con peticiones hasta que los datos estén ya revisados y se sepa que son correctos. Además, como se comentaba antes, esto parece más cómodo para el usuario, pues no tendrá que esperar a que el servidor le responda, sino que el propio navegador será el que le avise de que algo no es correcto.

Ante cualquier fallo detectado, JavaScript no permitirá que se envíen los datos y se le notificará al usuario cuál ha sido el error. Normalmente, el aviso se le mostrará al usuario en el propio campo erróneo, aunque, otra posibilidad sería mostrarle un mensaje de alerta que se verá en el siguiente apartado.

Un ejemplo de validación de un formulario sería este:

```
<form id="formAddUser" name="formPrueba" onsubmit="return validarDatos();" >
  <section>
    <label for="nombre">Nombre:</label>
    <div>
      <input id="nombre" type="text" name="nombre" maxlength="25" required value="" >
      <span></span>
    </div>
  </section>
  <section>
    <label for="dni">DNI:</label>
    <div>
      <input id="dni" type="text" name="dni" maxlength="9" required value="" >
      <span></span>
    </div>
  </section>
  <section>
    <input type="submit" value="Enviar datos" name="enviar">
  </section>
</form>
```

Ilustración 1: form.html

```
function validarDatos(){
  var form = document.forms["formPrueba"];
  if (form["nombre"].value.length < 5 ){
    alert("Nombre incorrecto");
  }else if (! form["dni"].value.match(/[0-9]{8}[A-Z]/)){
    alert("DNI incorrecto");
  }else{
    alert("Correcto");
    return true;
  }
  return false;
}
```

Ilustración 2: form.js

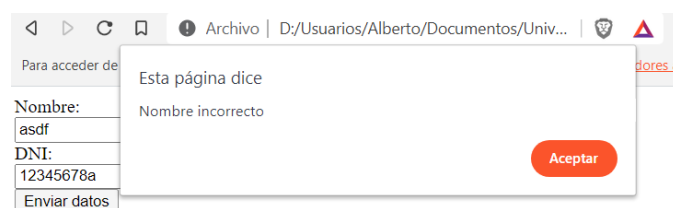


Ilustración 3: nombre incorrecto

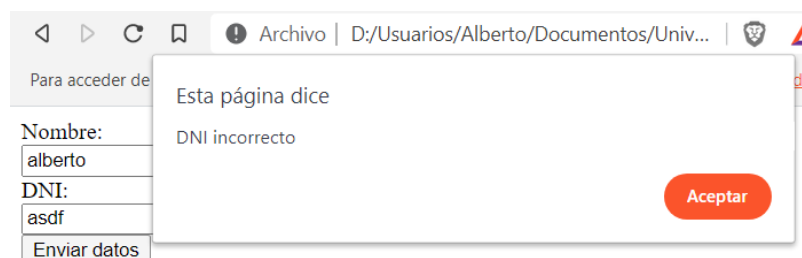


Ilustración 4: dni incorrecto

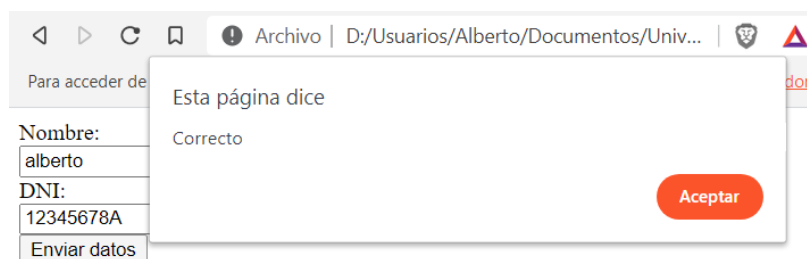


Ilustración 5: datos correctos

En este ejemplo, tras pulsar en aceptar en el botón de la Ilustración 5 se enviaría el formulario al servidor (en este caso no se envía nada al no haber puesto en el “form” ningún receptor). En el resto de casos, tras pulsar aceptar, el formulario no se enviaría, al haber retornado “false” en el apartado correspondiente de la función de JavaScript (Ilustración 2).

Enviar alertas y mensajes de confirmación

En las ilustraciones anteriores se han mostrado ya estas alertas, invocadas a través de la función alert(). Este mecanismo permite avisar de algo al usuario de forma que no pueda continuar en la página hasta que no atienda a dicha alerta, es decir, hasta que el usuario no pulse “Aceptar”, no podrá hacer nada en la página.

Otro tipo de alerta son las confirmaciones. En este caso, se pide al usuario que afirme o niegue algo en concreto. Por ejemplo, en el ejemplo anterior, podría solicitar al usuario que confirmara si realmente quiere enviar los datos, siempre que estos ya sean correctos:

```
function validarDatos(){
    var form = document.forms["formPrueba"];
    if (form["nombre"].value.length < 5 ){
        alert("Nombre incorrecto");
    }else if (! form["dni"].value.match(/[0-9]{8}[A-Z])){
        alert("DNI incorrecto");
    }else{
        if (confirm("Vas a enviar '" + form["nombre"].value + "' como nombre y '" + form["dni"].value + "' como DNI. ¿Estás seguro?")){
            return true;
        }
    }
    return false;
}
```

Ilustración 6: función confirm()

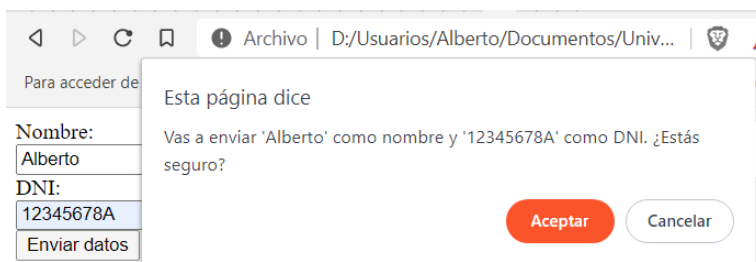


Ilustración 7: confirm en el navegador

En este último caso, si se pulsa el botón de aceptar se enviará el formulario, mientras que, si se pulsa cancelar, no se hará.

Modificar DOM

También es posible modificar lo que aparece en la página web, tanto a nivel de estilo como de contenido. Por ejemplo, podría implementarse un sistema de cambio de tema diurno a tema nocturno, podría insertarse un cierto texto introducido (o no) por el usuario al ocurrir un determinado evento, etc.

En el siguiente ejemplo, se ponen dos botones que quitan y ponen un recuadro al formulario y que quitan y ponen un texto en una etiqueta <div>, respectivamente:

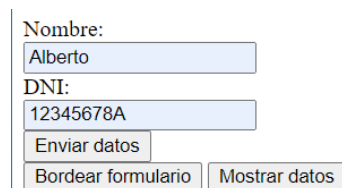
Un formulario web con dos campos de texto: 'Nombre:' con el valor 'Alberto' y 'DNI:' con el valor '12345678A'. Debajo de los campos hay tres botones: 'Enviar datos', 'Bordear formulario' y 'Mostrar datos'.

Ilustración 8: formulario antes de pulsar botones

El mismo formulario que en la ilustración 8, pero con un recuadro rojo que engloba los campos de texto y el botón 'Enviar datos'. Los campos ya están pre-llenados con 'Alberto' y '12345678A'.

Nombre: Alberto

DNI: 12345678A

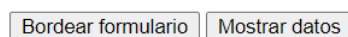
Dos botones: 'Bordear formulario' y 'Mostrar datos'.

Ilustración 9: Formulario tras pulsar botones

Requerir información al servidor sin salir de la página (Ajax)

Este último uso que se propone en la memoria proporciona la posibilidad de mostrar información en la página web haciendo una solicitud al servidor, pero sin tener que recargar toda la página. Por ejemplo, en el caso de registrar a un nuevo usuario, podría comprobarse, mientras continúa rellenando los demás campos, si el nombre de usuario no está ya en la base de datos, o incluir en la página información de manera más interactiva a la hora de mostrar algún tipo de dato que vaya requiriendo el usuario.

En el caso de ChartJS, puede hacerse uso de esta tecnología para modificar los gráficos teniendo en cuenta las solicitudes de datos que haga el usuario. Por ejemplo, en un gráfico de barras en el que se muestra un determinado set de datos, mediante Ajax, se podría solicitar más información al servidor para mostrarla en el gráfico, permitiendo comparar distintos sets de datos. Esto podría hacerse de esta manera para no tener que enviar toda la información en la carga de la página, sino que a medida que el usuario va requiriendo más información, esta se va enviando.

Librerías y frameworks JavaScript

En primer lugar, se debe dejar clara la diferencia entre estos dos conceptos, pues si bien ambos permiten facilitar ciertos procesos de código y, en principio, reducir la complejidad de este, son dos cosas distintas. Por un lado, una librería se centra más en resolver un aspecto concreto del código. Por ejemplo, ChartJS es una librería, ya que facilita mucho la creación de gráficos interactivos en las páginas web. Y por otro, un framework, es un concepto mucho más amplio, que lo que trata es de generar un entorno de programación con una cierta estructura que haga la programación más sencilla, al facilitar distintas funcionalidades de más alto nivel que un lenguaje en su forma base no tiene directamente implementadas.

En cuanto a los frameworks más utilizados actualmente (en ocasiones, según por quién, alguno de estos es considerado librería) están Angular, React, Vue, Ember, Backbone.js... Cada uno de ellos tiene sus peculiaridades, así que habrá que ver para cada tipo de proyecto cuál puede resolver más y de mejor forma los problemas a los que habrá que enfrentarse en dicho proyecto.

Por otro lado, al ser una librería un concepto mucho más concreto, hay miles de librerías publicadas en la web que resuelven o facilitan la resolución de cada uno de los miles de problemas a los que un desarrollador se enfrenta a la hora de crear un sitio web. [Aquí](#) pueden verse algunas de las más utilizadas en la actualidad, pero, por nombrar alguna, destacan ChartJS o D3.js para manejar datos y gráficos, wForms o LiveValidation para validar formularios, Anime para crear animaciones, typeface.js para modificar las fuentes, Date.js para manejar las fechas o Glimmer.js para hacer más “amigable” la interfaz de usuario de una aplicación.

ChartJS

Como se ha mencionado anteriormente, ChartJS es una librería de JavaScript para crear distintos tipos de gráficos, que además son “responsive” y se trata de una librería mantenida por la comunidad, al utilizar la licencia “open source”.

Instalación

En primer lugar, se debe instalar ChartJS mediante una de las posibles formas mencionadas en [su web](#), es decir, por medio de npm, CDN, jsDeliver o descargando el propio código de GitHub. En este caso, se utilizará el método CDN, es decir, incluyendo una etiqueta script en el head del documento html:

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.3.0/chart.min.js"
integrity="sha512-
yadYcDSJyQExcKhjKSQOkBKy2BLDoW6WnnGXCAkCoRlpHGpYuVuBqGObf3g/TdB86sSbss1
AOP4YlGSb6EKQPg==" crossorigin="anonymous" referrerpolicy="no-
referrer"></script>
```

En segundo lugar, se debe habilitar un espacio para el gráfico mediante la etiqueta <canvas>:

```
<div>
  <canvas id="canvas"></canvas>
</div>
```

Una vez hecho esto, pueden añadirse distintos tipos de gráfico en este canvas manejando todo desde Javascript.

Configuración básica

En ChartJS pueden generarse distintos tipos de gráfico de forma bastante sencilla. Además, la mayoría de ellos comparten una misma estructura, aunque después cada uno tenga alguna característica específica.

Básicamente, para generar un gráfico, hay que hacer una llamada al constructor de la clase Chart, la cual se descarga con el tag “script” del apartado anterior. Y a dicho constructor, se le debe pasar el tipo de gráfico (“line”, “bar”, “radar”, “pie”...), los datos que se quieren mostrar y un diccionario de opciones que puede contener o no instrucciones adicionales.

Sobre los datos a mostrar, se trata de un diccionario con una entrada de clave “labels” y valor una lista con las etiquetas de cada uno de los puntos en el eje de las X, y otra entrada de clave “datasets” y cuyo valor es una lista de diccionarios con la configuración de los distintos datasets, que contienen, de forma general, la “label” o etiqueta de ese dataset en concreto y los datos en formato de lista para el eje de las Y en el campo “data”. Además de estos campos “label” y “data”, incluyen otros referentes al color del gráfico, el tipo de líneas, etc., aunque depende de cada uno de los gráficos. Este sería un ejemplo de uno sencillo:

```
function graficoLinea() {
  //Función para generar lo necesario para el gráfico de líneas
  //se generan los datos
  let data = {}
  //etiquetas en x
  data["labels"] = ["a", "b", "c", "d", "e", "f", "g", "h"];
  data["datasets"] = [{
    label: "Dataset 1",
    data: [2, 5, 6, 8, 9, 6, 3, 2],
    fill: false,
    borderColor: "rgb(120, 25, 35)",
  }];
  dibujarGrafico("line", data);
}
```

```

}
function dibujarGrafico(type, data, options={}){
    let canvas = document.getElementById("canvas").getContext('2d');
    let chart = new Chart(canvas, {type:type, data:data,
options:options});
}

```

Distintos tipos de gráfico

Para que se entienda mejor, se han utilizado una serie de funciones accesorias que explico aquí:

- Dibujar gráfico: llama al constructor de Chart con los datos del gráfico.
- Obtener etiquetas: devuelve una lista de letras ordenadas del abecedario.
- Lista de números “random”: devuelve una lista de números aleatorios de la longitud pasada por parámetro.
- Color “random”: devuelve un color aleatorio del formato rgb(rrr,ggg,bbb), siendo rrr, ggg y bbb los valores correspondientes a rojo, verde y azul.

Dicho esto, paso a mostrar la codificación y resultado de algunos tipos de gráfico

Línea

```

function graficoLinea() {
    //se generan los datos
    let data = {}
    //etiquetas en x
    data["labels"] = obtenerEtiquetas();
    data["datasets"] = [{
        label: "Dataset 1",
        data: listaNumerosRandom(varsX),
        fill: false,
        borderColor: colorRandom()
    }]
    dibujarGrafico("line", data);
    tipoChart = "line";
}

```

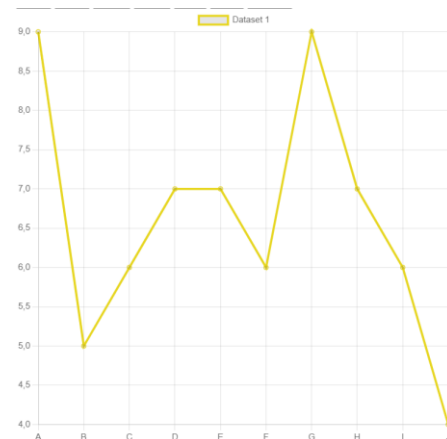


Ilustración 10: gráfico de líneas

Barra

```

function graficoBarra() {
    //se generan los datos
    let data = {}
    //etiquetas en x
    data["labels"] = obtenerEtiquetas();
    let listBorder= [];
    let listBack = [];
    for (let i=0;i<varsX;i++){
        let color = colorRandom();
        listBorder.push(color);
        listBack.push(color.substr(0,color.length-1) + ', 0.2');
    }
    data["datasets"] = [{
        label: "Dataset 1",
        data: listaNumerosRandom(varsY),
        borderColor: listBorder,
        backgroundColor: listBack,
        borderWidth: 1
    }]
    dibujarGrafico("bar", data);
    tipoChart = "bar";
}

```

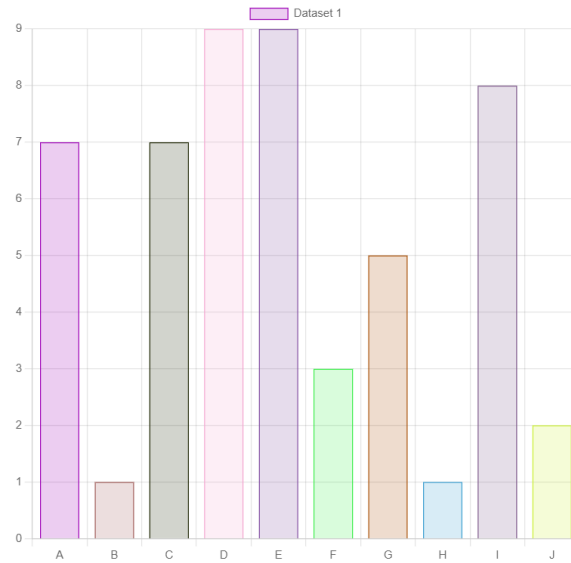



Ilustración 11: gráfico de barras

Radar

```
function graficoRadar() {
  //se generan los datos
  let data = {}
  //etiquetas en x
  data["labels"] = obtenerEtiquetas();
  let color = colorRandom();

  data["datasets"] = [{
    label: "Dataset 1",
    data: listaNumerosRandom(varsY),
    borderColor: color,
    backgroundColor: color.substring(0,color.length-1) + ",0.2)"
  }]
  dibujarGrafico("radar", data);
  tipoChart = "radar";
}
```

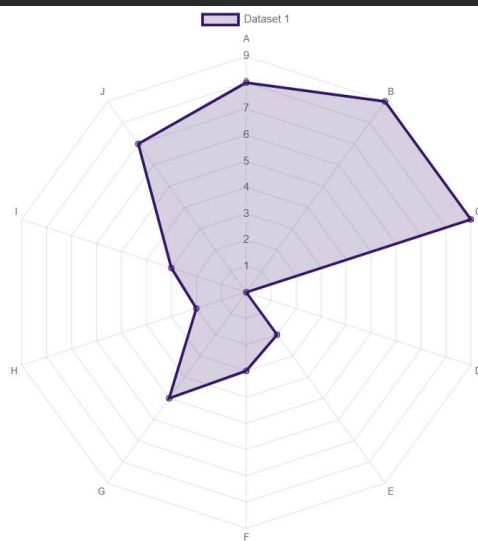


Ilustración 12: gráfico de radar

Tarta, Doughnut y Polar

```
function graficoTartaDoughnutPolar(tipo="pie") {
  //recibe pie, doughnut o polar
  //se generan los datos
  let data = {}
  //etiquetas en x
  data["labels"] = obtenerEtiquetas();
  let listBack = [];
```

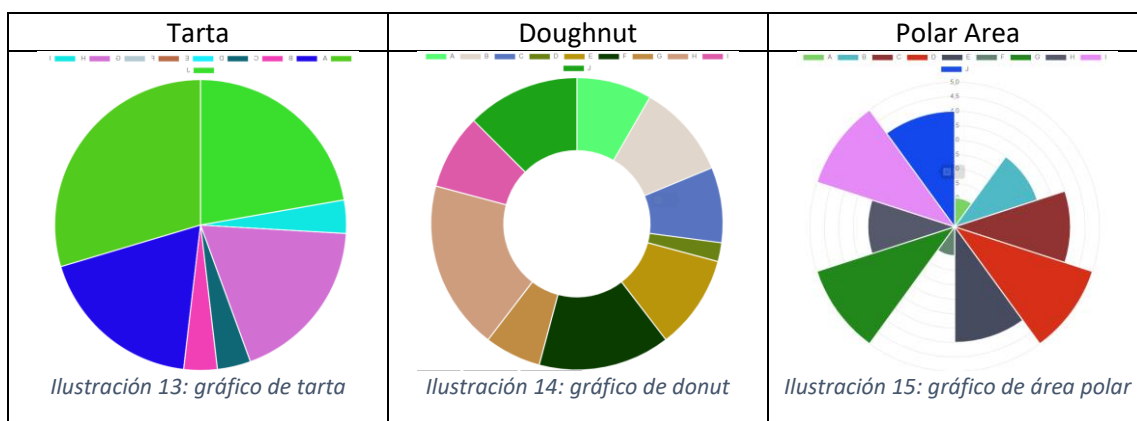
```

for (let i=0;i<varsX;i++){
    listBack.push(colorRandom());
}

data["datasets"] = [{
    label: "Dataset 1",
    data: listaNumerosRandom(varsY),
    backgroundColor: listBack
}]

dibujarGrafico(tipo, data);
tipoChart = tipo;
}

```



Opciones avanzadas

Como se ha visto en el apartado anterior, hay múltiples tipos de gráficos (no se muestran todos los posibles) para adaptar cada visualización al caso de uso para el cual sean necesarios. Y además de esto, los gráficos son altamente personalizables, ya que pueden modificarse un gran número de características, además de las vistas aquí, para lo que se utiliza el campo “options” del constructor de Chart.

Entre estas opciones, no solo están las modificaciones estéticas de colores, tipos de líneas o ubicación de los elementos dentro del canvas (arriba, abajo, izquierda, derecha, centro...), sino también qué ocurre cuando se pasa el cursor por encima del gráfico, qué se muestra en este caso, y muchas otras opciones de configuración que se dejan para un posible próximo trabajo.

Por último, comentar que el código utilizado para generar los ejemplos de la memoria puede consultarse en [mi github](#). Aquí puede verse alguna de estas opciones en la función “graficoOptions”, en el fichero chartjs.js. Además, en este fichero pueden verse también otras funciones que se han generado para responder a eventos como pulsaciones de botones para mostrar más datos, para aleatorizarlos o para generar nuevos dataset, así como para ir cambiando entre los distintos tipos de gráfico, lo cual tiene el siguiente resultado:

Gráficos ChartJS

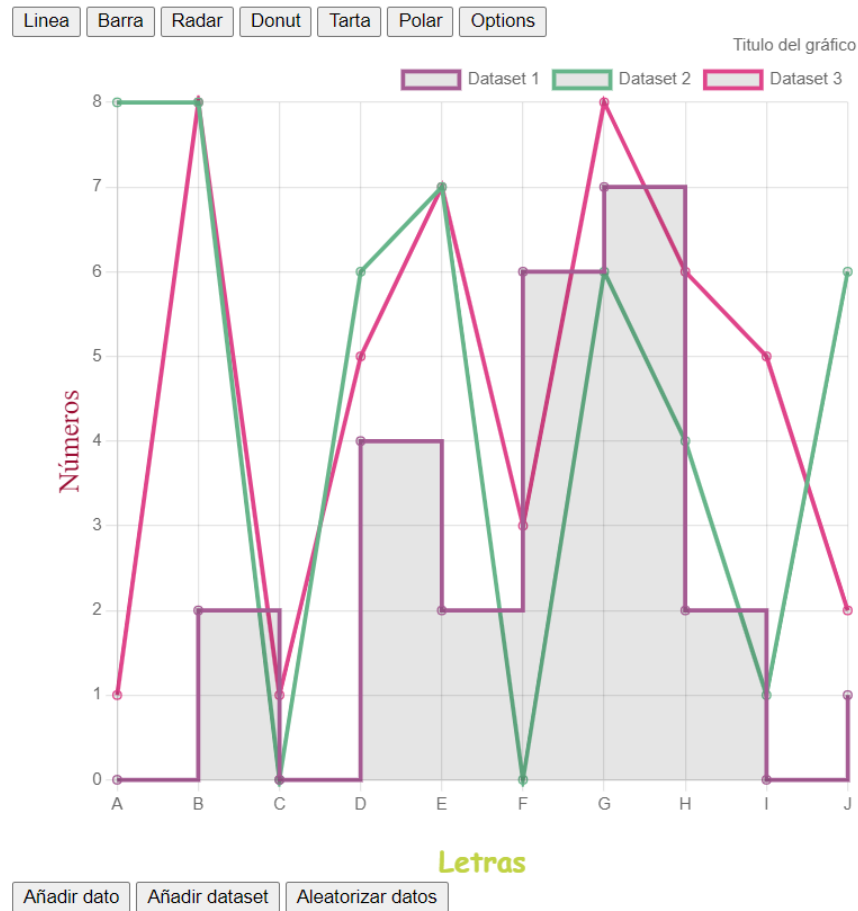


Ilustración 16: aplicación generada para los ejemplos

Bibliografía

Además de los sitios web enlazados a lo largo del documento, se utilizan los siguientes recursos:

- Sobre Javascript y su competencia: <https://www.quora.com/Why-is-JavaScript-the-only-client-side-language-available?share=1>
- Sobre ChartJS y sus distintos tipos de gráfico y configuraciones: <https://www.chartjs.org/docs/latest/>