

11.1 ¿QUÉ SON LAS APIs?

En varias ocasiones en este libro, hemos utilizado el término API (*Application Programming Interface*) explicando que es el conjunto de funciones, métodos, propiedades, objetos y componentes que permiten manejar un determinado software o componente. Así, por ejemplo el método **fetch**, los objetos **Response**, **Request** y **Headers**, con sus propiedades y métodos, forman, junto a otros elementos, la API conocida como **Fetch** que facilita la gestión de peticiones y respuesta de tipo AJAX. Incluso hay APIs mucho más extensas, como es el caso de la API DOM de JavaScript que permite manipular los elementos de las aplicaciones web y que está formada por cientos de métodos, propiedades y objetos.

Realmente las bases del lenguaje JavaScript ya están vistas y trabajadas en este libro, salvo algunos detalles avanzados o cuestiones de uso menos habitual. Lo que queremos decir es que lo fundamental para crear aplicaciones web en el lado del cliente es conocer el lenguaje JavaScript al máximo nivel posible, además hay que comprender y manipular el DOM perfectamente, es indispensable conocer cómo funciona AJAX para poder crear aplicaciones que utilizan datos y contenidos de servidores de Internet. A partir de aquí, JavaScript proporciona numerosas APIs más para realizar otro tipo de labores. Pero dependerá de nuestros intereses utilizar unas u otras.

Puede ser que algunas de las APIs que se comentan en esta unidad, no tengamos necesidad de tener que utilizarlas jamás. Hay APIs que son parte del estándar HTML5, otras las desarrollan terceros y no son estándares, aunque algunas de ellas son tan populares como las estándares.

Por ello, esta unidad tiene como principal pretensión entender cómo funcionan algunas de las más importantes, pero siempre con la idea de que si se necesita utilizar otras, el camino no va a ser complicado. La idea es entender cómo funcionan las APIs y como se aplican las más importantes, para así, si tenemos que aprender otras, no nos sea complicado.

Lo único que necesitamos conocer de las APIs son los métodos y elementos que proporcionan. Cuanto mejor las conoczamos, más grato será nuestro trabajo con ellas. En ese sentido es fundamental consultar su documentación.

11.2 APIs DE HTML5

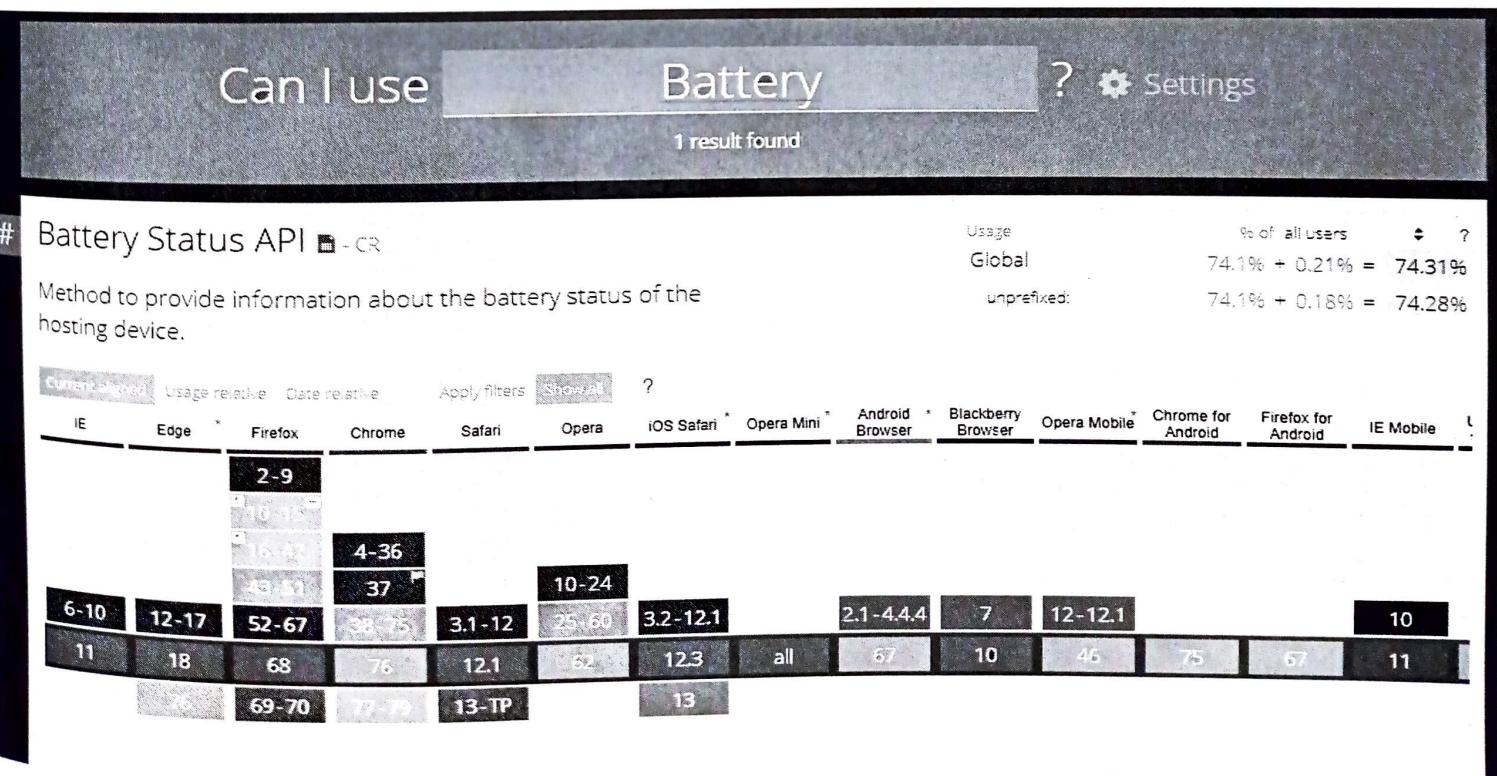
El estándar HTML5 cubre el lenguaje HTML, CSS y el propio JavaScript. A diferencia de la entidad ECMA que se encarga de estandarizar el lenguaje JavaScript en su norma ECMAScript, las entidades normalizadoras de HTML5 (fundamentalmente W3C y WHATWG) definen APIs creadas para JavaScript a fin de manipular aspectos de las aplicaciones web.

Ya hemos trabajado con APIs HTML5: la API Fetch y también el propio DOM, que es una API que proporciona objetos (**window**, **document**, etc.) y los métodos y propiedades para trabajar con ellos y que nos permiten manipular los elementos de un documento HTML y gestionar los eventos.

Ejemplos de ellas son:

- **Web Storage.** Que permite el almacenamiento de datos en el navegador. Su finalidad es no depender de las cookies para realizar esta labor.

- **Web Workers.** API para la creación de tareas pesadas que se ejecutan en segundo plano y que, en primer plano, ralentizarían el trabajo.
- **Geolocation.** API que permite gestionar coordenadas de localización (longitud y latitud) con el fin de utilizarlas en aplicaciones web.
- **Web Sockets.** Permite la creación de sockets de red para la comunicación directa entre el cliente y un servidor.
- **API Canvas.** Para la manipulación del elemento canvas de HTML5 que permite la creación de gráficos en dos dimensiones.
- **Web GL.** API que incorpora las capacidades de **Open GL** para crear gráficos y animaciones en tres dimensiones en los elementos **canvas**.
- **FileSystem API.** API para poder utilizar desde JavaScript archivos locales.
- **Application Cache.** Elemento fundamental para la creación de aplicaciones web que permitan el trabajo offline.
- **Media API.** Que permite manipular elementos de vídeo y audio en una aplicación web.
- **Text Track API.** Que permite manipular subtítulos para componentes de vídeo y audio.
- **Drag and Drop.** Para facilitar la programación de componentes arrastrables.
- **FullScreen.** API para poder manipular las aplicaciones web en estado de "pantalla completa".



- **Battery Status.** API para poder obtener información de la batería del dispositivo.

Hay muchas más. No todas están soportadas por todos los navegadores, pero muchas sí lo están y se pueden utilizar con total tranquilidad. La página <https://caniuse.com/> nos permite saber la compatibilidad con los navegadores de las principales APIs de HTML5. De este modo podremos decidir si usarlas o no en función de su grado de implantación real.

Año a año, se van añadiendo nuevas APIs que proporcionan nuevas funcionalidades y, también, se modifican las APIs existentes para adaptarlas a las nuevas capacidades del lenguaje JavaScript o para modernizar alguno de sus aspectos. Conviene documentarse de vez en cuando sobre estos cambios.

11.3 API PARA CANVAS

11.3.1 ¿QUÉ ES CANVAS?

Canvas es un elemento de HTML5 que permite crear gráficos y animaciones utilizando comandos de JavaScript. El elemento canvas de HTML es un marco que se posiciona en el sitio que deseamos y que podemos modificar de tamaño:

```
<canvas id="lienzo" width="500" height="300"></canvas>
```

Desde JavaScript podemos acceder a este componente y, a través de los métodos que nos proporciona la API de Canvas, realizar creaciones gráficas en el espacio que proporciona el elemento **canvas**.

Este elemento tuvo un éxito muy grande desde su aparición en el estándar. Actualmente se sigue usando mucho, pero tiene bastante competencia con el éxito que también están teniendo los gráficos **SVG**.

11.3.2 EMPEZAR A TRABAJAR CON CANVAS

Como hemos comentado, en primer lugar debemos obtener una referencia al elemento **canvas** desde JavaScript:

```
const lienzo=document.getElementById("lienzo");
```

Después debemos obtener el contexto del elemento.

```
const ctx=lienzo.getContext("2d");
```

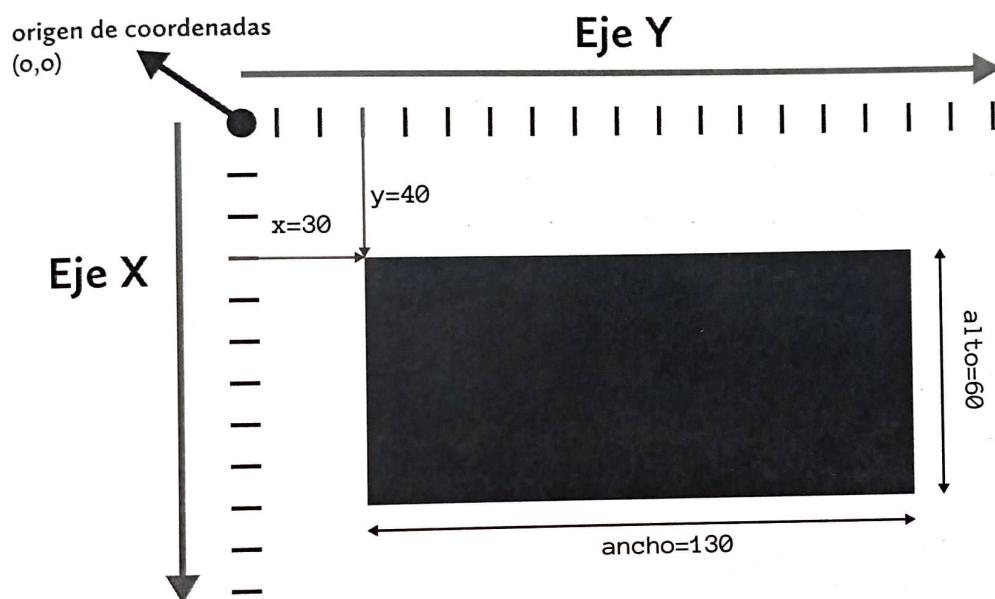
El contexto es el objeto que permite dibujar en el lienzo. Proporciona todos los métodos y propiedades de dibujo. El método **getContext**, presente en los elementos de tipo canvas, es el encargado de obtener el contexto. A este método se le pasa un string que puede contener los valores **2d** o **3d**. Normalmente se usa el valor **2d** porque es el que usa para gráficos en dos dimensiones, que son los habituales. El valor **3d** permite el dibujo de gráficos 3d con ayuda de la librería **WebGL**.

A partir de este momento, se pueden utilizar los métodos que permiten elegir las propiedades de trazo y relleno de los objetos que dibujemos y los métodos que permiten dibujar dichos objetos.

En muchas ocasiones tenemos que indicar coordenadas y, por ello, hay que entender como funciona el modelo de coordenadas de canvas.

El origen de coordenadas está en la esquina superior izquierda. Se llama eje **x** al horizontal e **y** al vertical. Cuando se dan posiciones de puntos, se indica siempre primero la coordenada **x** y luego la **y**. La dirección de las coordenadas positivas va hacia abajo (eje **x**) y a la derecha (eje **y**).

El tamaño en píxeles de las coordenadas del **canvas** lo ajustan las propiedades **width** y **height** de la etiqueta **canvas** de HTML.



```
contexto.fillRect(30, 40, 130, 60);
```

Figura 11.2: Funcionamiento del sistema de coordenadas de canvas

En la imagen anterior se muestra como se coloca un rectángulo en un área **canvas** (señalado en gris claro) mediante el método **fillRect**. Los dos primeros valores indican la posición de la esquina superior izquierda del mismo (coordenadas 30,40) el siguiente la anchura(130) y el último, la altura (60).

11.3.3 DIBUJO DE FORMAS

11.3.3.1 PROPIEDADES DE LAS FORMAS

Una forma es una figura geométrica pintada sobre el lienzo. Las formas tienen un color de fondo o relleno y un borde. El relleno se indica con un color y el borde tiene color y también grosor. Para indicar el color de relleno actual se usa la propiedad **fillStyle**. A esta propiedad se la puede indicar un color al estilo de CSS, por ejemplo "**blue**" o "**#0000FF**". La propiedad **strokeStyle** es similar y sirve para indicar el color del borde. Hay otra propiedad llamado **lineWidth** que permite indicar la anchura del borde.

Una vez indicadas estas tres propiedades, las figuras dibujadas utilizarán esos colores y grosores de línea. Si no los especificamos, se utilizarán los valores por defecto configurados en el navegador del usuario.

11.3.3.2 RECTÁNGULOS

Canvas solo dispone de una forma básica, el rectángulo. Se dispone de dos métodos para dibujar rectángulos:

- **fillRect**. Dibuja un rectángulo sólido. Para hacerlo se indica la coordenada superior izquierda del rectángulo y, la anchura y la altura del mismo.
- **strokeRect**. Dibuja un rectángulo transparente y pinta el borde.

Ejemplo:

```
<canvas id="lienzo" width="500" height="300"></canvas>
<script>
const lienzo=document.getElementById("lienzo");
const contexto=lienzo.getContext("2d");
contexto.fillStyle="red";
contexto.lineWidth=2;
contexto.strokeStyle="gray";
contexto.fillRect(10,10,150,100);
contexto.strokeRect(30,30,150,100);
</script>
```

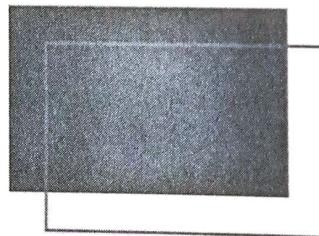


Figura 11.3: Rectángulos dibujados con los métodos **fillRect** y **strokeRect**.

11.3.3.3 TRAZOS

Un trazo es una serie de dibujos de líneas que forman una figura común. Los trazos deben comenzar invocando el método **beginPath**. Este método sirve para marcar dónde comienza el trazo. Desde ese momento todos los comandos se entienden que van dirigidos a formar el trazo.

El trazo se finaliza invocando, sin argumentos, al método **closePath**, el cual cierra el trazo. Tras esa instrucción las siguientes acciones se entiende que ya no van dirigidas a ese trazo. El trazo se dibuja realmente invocando al método **stroke** (dibuja el contorno) o a **fill** (pinta el relleno). Tanto **stroke** como **fill** usan la configuración de relleno y trazo actual establecida por los métodos **strokeStyle**, **fillStyle** y **lineWidth** comentados anteriormente.

Tras la instrucción **beginPath**, los métodos **moveTo** y **LineTo** permiten dibujar trazos poligonales. El primero (**moveTo**) permite colocar el **cursor** en una coordenada concreta. El segundo

(`lineTo`) marcará una línea desde la última posición del cursor a la posición indicada por los parámetros de `lineTo`. Los parámetros de ambos métodos son una coordenada "x" y otra "y".

Ejemplo:

```
const lienzo=document.getElementById("lienzo");
const ctx=lienzo.getContext("2d");
ctx.beginPath();
ctx.lineWidth=2;
ctx.strokeStyle="blue";
ctx.moveTo(50,10);
ctx.lineTo(100,60);
ctx.lineTo(50,110);
ctx.lineTo(0,60);
ctx.lineTo(50,10);
ctx.stroke();
ctx.closePath();
```

El resultado sería:

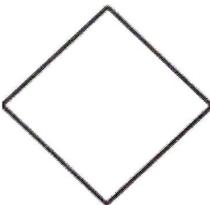


Figura 11.4: Rombo dibujado con los métodos de creación de trazos `lineTo` y `moveTo`.

Hay un método que dibuja trazos rectangulares. Se llama `rect`. Los parámetros son los habituales en el relleno de rectángulos: coordenadas de la esquina superior izquierda (x e y), anchura y altura.

11.3.3.4 ARCOS

El método `arc` también se usa para crear trazos. De hecho se suele combinar con los anteriores. Tiene como parámetros: las coordenadas (x e y) en las que se debe situar el centro del arco, el tamaño del radio, el ángulo inicial y el ángulo final. Los ángulos se miden en radianes, una medida muy matemática que se usa mucho en ciencias matemáticas y en programación. Cabe decir que, en esa unidad, dos veces PI equivale a 360 grados. 90 grados serían PI/2. Hay un último parámetro que indica, con un valor booleano, la dirección del arco: el valor `true` indica que se ha de dibujar en la misma dirección que las agujas del reloj.

Ejemplo:

```
const lienzo=document.getElementById("lienzo");
const ctx=lienzo.getContext("2d");
ctx.beginPath();
ctx.lineWidth=2;
```

```

ctx.strokeStyle="blue";
ctx.beginPath();
ctx.moveTo(50,10);
ctx.lineTo(100,10);
ctx.lineTo(100,110);
ctx.lineTo(50,110);
ctx.stroke();
ctx.closePath();
ctx.beginPath();
ctx.arc(50,60,50,Math.PI*3/2,Math.PI/2,true);
ctx.stroke();
ctx.closePath();

```

El resultado es:

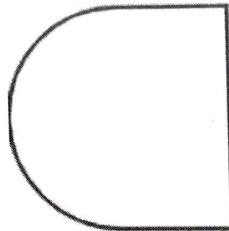


Figura 11.5: Figura conseguida con los métodos `lineTo` y `arc`.

11.3.4 DIBUJO DE TEXTO

La propiedad **font** permite indicar las propiedades del texto que vayamos a dibujar en el canvas. Las propiedades a indicar se separan con espacios (al estilo de la propiedad **font** de CSS dentro de un string).

El método **fillText** dibuja y rellena un texto cuyo contenido es el primer parámetro. Después se indican dos coordenadas que serán la posición inferior izquierda para la línea base del texto.

```

ctx.lineWidth=2;
ctx.strokeStyle="black";
ctx.beginPath();
ctx.moveTo(50,50);
ctx.lineTo(200,50);
ctx.stroke();
ctx.strokeStyle="red";
ctx.font="bold 30px sans-serif";
ctx.fillText("Prueba",50,50);

```

Prueba

Figura 11.6: Texto y línea dibujados con `lineTo` y `fillText`.

11.3.5 DIBUJAR IMÁGENES

Una de las opciones más potentes de canvas es poder colocar una imagen (gif, jpg, etc.) en el lienzo. En este sentido disponemos de un constructor de objeto llamado **Image** que crea un objeto de imagen equivalente al elemento **img** de HTML. Podemos modificar las propiedades habituales de las imágenes, por ejemplo, la propiedad **src**.

Canvas proporciona un método llamado **drawImage** que, como primer parámetro, recibe un objeto de imagen y después dos coordenadas que marcarán dónde quedará la esquina superior izquierda de la imagen. Además, admite dos parámetros más con los que podemos modificar la anchura y altura de la imagen.

```
const img=new Image();
img.src="https://img.icons8.com/material-outlined/96/000000/dog.png";
img.addEventListener("load", (ev)=>{
    ctx.lineWidth=2;
    ctx.strokeStyle="black";
    //cuadrado contenedor
    ctx.strokeRect(200,150,96,96);
    //dibujo de la imagen
    ctx.drawImage(img,200,150);
});
```

En el código, las instrucciones de dibujo se colocan en la función callback del evento load de la imagen porque hay que asegurar que el código se ejecuta tras la carga de dicha imagen.



Figura 11.7: Imagen colocada con **drawImage**. El recuadro lo consigue **strokeRect**.

11.3.6 BORRAR

El método **clearRect** dibuja un área rectangular sobre el canvas, con la intención de eliminar todo lo que hay dentro. Es decir, borra esa área y la deja con color transparente. Los parámetros son los habituales para indicar rectángulos:

```
const lienzo=document.getElementById("lienzo");
const contexto=lienzo.getContext("2d");
contexto.fillStyle="red";
contexto.lineWidth=2;
contexto.strokeStyle="gray";
contexto.fillRect(0,0,200,200);
contexto.fill();
```

```
contexto.stroke();
contexto.clearRect(100,100,200,200);
```

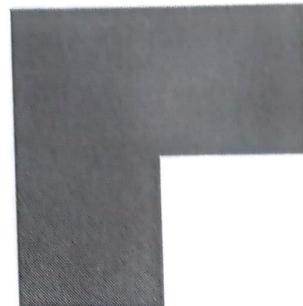


Figura 11.8: Forma lograda con el apoyo de clearRect

11.3.7 CONCLUSIONES SOBRE CANVAS

Hay muchas más funciones y posibilidades para crear gráficos en Canvas. Evidentemente, crear un gráfico complicado usando estos métodos es difícil, pero combinado con otras utilidades de JavaScript (temporizadores, eventos, etc.) con formas muy sencillas se logran efectos muy llamativos e interesantes.

Hay programas gráficos profesionales que permiten dibujar en un entorno de trabajo más amigable y después convertir el resultado a formato canvas. Esto es, por ejemplo, lo que permite el software ***Adobe Animate and Mobile Device Packaging*** (antiguo ***Adobe Flash***), se crean escenarios utilizando las potentes herramientas de este software y luego se le pide que cree un archivo con las instrucciones JavaScript que lo conseguirían dibujar en un elemento canvas.

Otra cuestión es que existen APIs de terceros para trabajar con gráficos avanzados en las aplicaciones mucho más avanzadas y fáciles que Canvas(***Tween, paper.js, Raphael.js***, etc.). Aunque, como siempre ocurre al usar librerías externas, siempre es más veloz para el navegador usar las APIs nativas.