ÍNDICE DE CONTENIDO

1.- COOKIES

Puesto que HTTP es un protocolo sin estado, cada vez que se pide una página en el servidor, el servidor no "recuerda". Esto, como puede imaginar, supone un grave inconveniente, ya que no es posible, por ejemplo, cambiar de página sin perder toda la información, como puede ser:

- Estar autenticados en un sitio web
- La cesta de la compra en un negocio de venta por internet
- Las preferencias seleccionadas al visitar un sitio web

Tampoco se puede acceder a los siguientes servicios:

- Realizar el seguimiento de las páginas visitadas para generar analíticas
- Mostrar vídeos (Youtube)
- Mostrar mapas (Google Maps)
- Mostrar anuncios publicitarios
- Servicios de chat

La solución que se dio a este problema fue la invención de las cookies que son unos pequeños fragmentos de información que el navegador guarda en archivos y se envían junto con la cabecera al servidor; de modo que se puede simular que el servidor nos recuerda, todo que en realidad siempre se trata cada petición como si fuera nueva.

Como estas cookies pueden ser leídas y escritas desde JavaScript, se pueden utilizarse para otros propósitos que no están relacionados con el servidor; por ejemplo, para guardar las preferencias de un usuario al visitar una página. Se debe tener en cuenta que el tamaño máximo para cada cookie es de 4.095 bytes.

Con HTML5 se añadió una nueva Web API para almacenar información en navegador llamada Web Storage que, a diferencia de las cookies, no se envían nunca en el servidor.

Así pues, si necesitamos compartir la información guardada con el servidor, haremos Sin embargo, si utiliza cookies, si se tiene que acceder sólo localmente, se recomienda utilizar Web Storage, ya que acepta hasta 5 MB por dominio y se ahorra ancho de banda porque no se envía esta información al servidor (al contrario de lo que ocurre con las cookies).

Si se desactiva el uso de cookies en el navegador, quedan inhabilitadas tanto la Web Storage como las cookies.

Las cookies pueden visualizarse desde las herramientas de desarrollador de los navegadores, pero su ubicación cambia según el navegador con el que trabaja.

En Google Chrome puede encontrarlas en la pestaña Resources, en el desplegable Cookies del panel izquierdo.

En cambio, si utilice Mozilla Firefox, las puede encontrar en la pestaña Almacenamiento, en el desplegable Cookies del panel izquierdo.

1.1.- Ley de cookies

Antes de adentrarnos en la gestión de cookies debemos hablar de la normativa que en regula el uso, ya que si no se informa adecuadamente al usuario que las utilizamos y con qué finalidad, podemos ser sancionados. En estos materiales no trataremos el tema en profundidad, pero es importante conocerlas y saber que siempre que trabajemos con cookies tendremos que aplicar esta ley.

A la hora de trabajar con cookies se debe tener muy presente la legislación vigente, que nos obliga a informar al usuario que nuestro sitio web utiliza cookies y con qué fines, y con un enlace que explique cómo puede desactivarlas en su navegador.

En la práctica es muy difícil encontrar un sitio web que no utilice cookies, ya que son imprescindibles incluso para generar analíticas. Por esta razón puede dar por descontado que deberá informar al usuario que su sitio web utiliza cookies y respetar la normativa.

Se debe tener en cuenta también que si los datos guardados en las cookies son de carácter personal, debe aplicarse también la Ley orgánica de protección de datos (LOPD).

1.2.- Utilización de las cookies

El funcionamiento de las cookies es muy sencillo, pero utilizarlas no lo es tanto. Se accede a través de la propiedad document.cookie, tanto para recuperarlas como para guardarlas. Puede verlo en el siguiente ejemplo:

```
<script>
 // Añadimos una nueva cookie
    document.cookie="sede=Alicante";
    document.cookie="paf=1";
 // Recuperamos todas las cookies
    let galleta = document.cookie;
 // Las mostramos en la consola
    console.log(galleta);
</script>
```

Si comprueba la consola, verá el contenido de su cookie. En caso de probarlo en CodePen, verá también las cookies añadidas por CodePen separadas por un ; unas de otras: sede=Barcelona; paf=1; grid_layout=list; _gat=1; _ga=GA1.2.197952496.1453793789

Lo primero que ve en este ejemplo es cómo se guarda una información a una galleta; siempre debe ser con el siguiente formato: clave=valor, y si existen más cookies, estarán separadas con un ;.

Seguramente le ha extrañado este fragmento de código:

```
// Añadimos una nueva cookie
   document.cookie="sede=Alicante";
   document.cookie="paf=1";
```

Se podría esperar que al sobreescribir el valor de la propiedad document.cookie, este valor fuera paf=1 y su valor=Alicante fuera eliminado. Esto no ocurre, porque esta propiedad tiene unos setter y getter nativos que modifican su comportamiento y, por tanto, no es lo mismo el que se escribe y lo que se lee.

Intentemos ahora guardar múltiples cookies:

```
<script>
 // Añadimos nuevas cookies
    document.cookie="sede=Alicante";
    document.cookie="sede=Alcoy";
    document.cookie="sede=Elche";
    document.cookie="sede=Benidorm";
 // Recuperamos todas las cookies
    let galletas = document.cookie;
 // Las mostramos a la consola
    console.log(galletas);
</script>
```

Como puede apreciar, sólo se ha guardado el valor sede=Benidorm. En realidad se han guardado todos, pero los siguientes los han sobreescrito, ya que sólo puede haber uno valor asociado a cada clave.

Una posible solución sería guardar los datos como un array, pero las cookies sólo pueden guardar cadenas de texto, y por tanto, si necesitamos un comportamiento más avanzado, debemos implementarlo nosotros mismos o utilizar alguna librería externa.

Después de entender cómo guardar y recuperar las cookies, se nos presenta un problema nuevo: ¿cómo lo hacemos para recuperar sólo la galleta que nos interesa y no todas? Desgraciadamente ya ha visto todo el soporte proporcionado por la especificación, es decir: cómo añadir una cookie, cómo sobreescribir su valor y cómo recuperarlas todas. Si queremos recuperar sólo una, debemos implementar nuestra propia solución, por ejemplo:

```
<script>
 // Añadimos nuevas cookies
    document.cookie="sede=Alicante";
    document.cookie = "paf=1";
 // Extraemos la cookie correspondiente a la sede
    let patron = new RegExp("(?:seu)=(.+?)(?=;|$)", "g"),
        sede = patro.exec(document.cookie)[1];
 // Las mostramos a la consola
    console.log(sede);
</script>
```

Lo primero que hacemos es definir un patrón que nos capturará el valor de la galleta con el su nombre, ya continuación lo ejecutamos. El resultado de ejecutar la expresión regular es un array que contiene en la primera posición (índice 0) el texto coincidente completo y en las siguientes posiciones el contenido de los grupos de captura ignorando el primero, ya que tiene el formato de grupo no capturable: (?:). Es decir, en el índice 1 se encuentra el contenido del grupo que se encuentra a continuación del signo igual, ya que es el primer grupo de captura válido y, por consiguiente, se corresponde con el valor de la sede.

Todo esto es bastante enrevesado, así que, para simplificarlo, podemos crear una función que nos facilitará la tarea, como en este ejemplo:

```
<script>
 // Añadimos nuevas cookies
    document.cookie="sede=Alicante";
    document.cookie = "paf=1";
 // Recuperamos las cookies
    let paf = getCookie('paf'),
    sede = getCookie('sede');
 // Las mostramos a la consola
    console.log("Todas las cookies:", document.cookie);
    console.log("sede:",sede);
    console.log(PAF:",paf);
 // Extraemos la cookie correspondiente a la sede
    Function getCookie(key) {
        let patron = new RegExp("(?:seu)=(.+?)(?=;|$)", "g");
        return patron.exec(document.cookie)[1];
    }
</script>
```

Como puede comprobar, al utilizar una función que encapsule la lógica para recuperar una galleta, se simplifica mucho su utilización; sólo debemos invocar getCookie() y pasar el nombre de la clave que queremos recuperar.

Aparte de este método -más simple- para guardar una galleta, las cookies admiten otras opciones para limitar su uso y caducidad; son las siguientes:

a) **path:** por ejemplo /, /actualidad. Si no se especifica en la cookie, se podrá acceder desde todas las páginas del sitio web.

b) **domain:** por ejemplo ejemplo.com, subdominio.ejemplo.com. Si se especifica el dominio principal, también se podrá leer desde los subdominios; si no se especifica, sólo se podrá leer desde el dominio principal.

c) **max-age:** duración máxima en segundos hasta que la galleta caduque.

d) **expiras:** fecha en formato GMT en el que expirará la cookie. Si no se especifica, la cookie caduca cuando finaliza la sesión (generalmente cuando se cierra la pestaña concreta o el navegador)

e) **secure:** la cookie sólo se podrá transmitir sobre protocolos seguros como HTTPS. Este tipo de cookies no necesita valor. Todas las cookies enviadas a través de HTTPS son automáticamente seguras.

Aunque existe una opción secure, las cookies son inherentemente inseguras, y no debe utilizarlas nunca para guardar información sensible ni confidencial.

Cada opción y su valor irá separado por un punto y coma (;) y un espacio en blanco, separando los pares opción y valor por un signo igual (=).

Aprovechando que ya conocemos todas las opciones posibles, implementaremos la función setCookie():

```
<script>
function setCookie(key, value, options) {
  let cookie = encodeURIComponent(key) + '=' + encodeURIComponent(value);
  if (options) {
    for (option in options) {
    if (option === 'secure') {
      cookie += '; ' + encodeURIComponent(option);
    } else {
      cookie += '; ' + encodeURIComponent(option) + '=' + encodeURIComponent(
      options[option]);
        }
    }
  }
  document.cookie = cookie;
}

function getCookie(key) {
  let patron = new RegExp("(?:" + key + ")=(.+?)(?=;|$)", "g");
  return patron.exec(document.cookie)[1];
}
// Añadimos una nueva cookie
  setCookie('sede', 'Castalla', {
   'max-age': 300,
    path: '/',
    domain: 'codepen.io'
  });

// Mostramos los datos por consola
  console.log("Todas las cookies:", document.cookie);
  console.log("Sede:", getCookie('sede'));
</script>
```

Fíjese que, para pasar las opciones de las cookies, hemos utilizado un objeto literal de JavaScript, de esta forma el código queda mucho más limpio y compacto. La propiedad max-age, al incluir el signo -, debe ponerse entre comillas.

Si intenta pasar el parámetro secure no le funcionará porque requiere que la conexión se haga sobre HTTPS, y ni el ejemplo en local ni en CodePen corren sobre este protocolo.

Aunque la especificación no requiere realizar la codificación completa de los contenidos de la cookie (sólo los espacios, , y ; deben ser codificados) es más simple codificar todos los valores y claves utilizando la función encodeUriComponent(). De ésta modo nos aseguramos que los valores guardados serán correctos.

1.3.- Tiempo y duración de las cookies

Aunque nuestra función para guardar cookies rule y contemple todas las opciones disponibles, todavía podemos tener problemas con un apartado: establecer la fecha en la que queremos que expire.

La especificación nos dice que el formato debe ser el siguiente: Wdy, DD-Mon-YYYY HH:MM:SS GMT como por ejemplo: Sat, 02 May 2009 23:38:25 GMT. Pero si utilizamos el objeto Date de JavaScript lo que nos devuelve será parecido a esto:

```
Date.now();
// 1456167356375
```

El objeto Date nos devuelve la fecha en milisegundos. Afortunadamente, dispone de métodos para poder modificar esa fecha; veamos algunos ejemplos:

```
<script>
  let ahora = new Date(),
  mes,
  proximMes,
  navidad;

  console.log("La fecha de hoy en UTC", ara.toUTCString());

  mes = ahora.getMonth();
  console.log("¿Cuál es el mes actual?", mes);

  proximMes = (mes + 1) % 12;
  ahora.setMonth(proximMes);
  console.log("¿Cuál es la fecha correspondiente al próximo mes?", ara.toUTCString()
);

   navidad = new Date(2016, 11, 25);
   console.log("¿Qué día es Navidad en 2016?", nadal.toUTCString());

  console.log("Desfase horario por nuestra localización (en minutos):", nadal.getTimezoneOffset()
)
</script>
```

El resultado de invocar el método toUTCString() produce una fecha exactamente con el formato que necesitamos, como esta: Tue, 22 Mar 2016 19:08:01 GMT. Fíjese que, en el último ejemplo, creamos una fecha en concreto, pero el resultado puede ser inesperado, ya que:

Enero corresponde al mes 0, y por tanto diciembre es el mes 11.

La fecha mostrada será esta: Sat, 24 Dec 2016 23:00:00 GMT

El desfase horario será de -60 minutos, esto explica por qué la fecha mostrada parece errónea por una hora. Cuando se crea el objeto Date se hace servir el uso horario del cliente, que en nuestro caso es GMT+1, y al convertirla en GMT se descuenta esa hora.

Trabajar con fechas añade una complicación extra a la creación de cookies. Por esta razón es más recomendable utilizar la opción max-age, que nos permite especificar la edad máxima que puede alcanzar la cookie, expresada en segundos.

2.- WEB STORAGE

2.1.- Utilización de Web Storage

Esta Web API permite almacenar datos en el ordenador cliente con más posibilidades de que las galletas. Distingue dos formas de realizar este almacenamiento:

- **LocalStorage**: los datos no expiran nunca y, por tanto, se requiere borrado manual o por programa
- **SessionStorage**: los datos se guardan sólo por una sesión y, al cerrar la ventana del navegador se borra.

Aparte de lo anterior, el funcionamiento es idéntico a ambos tipos de Web Storage.

LocalStorage comparte la información entre todas las páginas del mismo dominio, en cambio, SessionStorage es diferente para cada ventana; por lo que si tenemos tres ventanas abiertas con la misma página, la información del SessionStorage será diferente para cada una, mientras que la del LocalStorage será la misma.

A diferencia del uso de cookies, esta Web API ofrece una interfaz muy clara para interactuar con el almacén de datos:

- **Storage.setItem(key, value)**: para guardar un valor con la clave especificada.
- **Storage.getItem(key)**: para recuperar el valor correspondiente a la clave.
- **Storage.removeItem(key)**: para eliminar el elemento correspondiente a la clave del almacén.
- **Storage.clear()**: elimina todas las claves del almacén.

Veamos cómo funcionan todos estos métodos:

```
<script>
        localStorage.setItem('sedes', ['Alicante', 'Elche', 'Alcoy', 'Villena']);
        localStorage.setItem('sede seleccionada', 'Barcelona');
        localStorage.setItem('paf', 1);
        localStorage.setItem('confirmar', false);

        console.log("comprobar que se han creado:"); console.log("sedes:", localStorage.getItem('sedes'));
        console.log("seleccionada:", localStorage.getItem('sede seleccionada'));
        console.log("paf:", localStorage.getItem('paf'));
        console.log("confirmar:", localStorage.getItem('confirmar'));

        console.log("Eliminar la sede seleccionada.")
        localStorage.removeItem('sede seleccionada');

        console.log("comprobar el valor de la sede seleccionada:", localStorage.getItem('sede seleccionada'));

        console.log("Eliminar todos los valores");
        localStorage.clear();

        console.log("comprobar que se han eliminado:")
        console.log("sedes:", localStorage.getItem('sedes'));
        console.log("seleccionada:", localStorage.getItem('sede seleccionada'));
        console.log("paf:", localStorage.getItem('paf'));
        console.log("confirmar:", localStorage.getItem('confirmar'));
</script>
```

Para utilizar SessionStorage en lugar del LocalStorage, sólo tenemos que cambiar localStorage por sessionStorage.

Como puede ver, a diferencia de las cookies, los valores almacenados pueden ser de diferentes tipos: enteros, cadenas, booleanos e incluso arrays (solo habrá que hacer servir String.split() para recuperar el array).

No es necesario crear ningún objeto ni configurar nada para acceder a los almacenes de datos, son accesibles directamente a través de los objetos globales localStorage y sessionStorage que nos proporciona el navegador, y la información se guarda automáticamente asociada al dominio en el que se ha creado.

La persistencia de los datos se basa en el tipo de almacén: si utilizamos el LocalStorage estos datos persistirán hasta que los borremos nosotros mismos vía código o cuando el usuario borre los datos de navegación; en cambio, si utilizamos el SessionStorage , todos los datos se borrarán tan pronto como se cierre la ventana del navegador.

2.2. Local Storage y Session Storage

LocalStorage y sessionStorage son propiedades de HTML5 (web storage), que permiten almacenar datos en nuestro navegador web. De manera muy similar a como lo hacen las cookies.

- **Local Storage**: Guarda información que permanecerá almacenada por tiempo indefinido; sin importar que el navegador se cierre.
- **Session Storage**: Almacena los datos de una sesión y éstos se eliminan cuando el navegador se cierra.

Las características de *Local Storage* y *Session Storage* son:

a) Permiten almacenar entre 5MB y 10MB de información; incluyendo texto y multimedia.
b) La información está almacenada en la computadora del cliente y NO es enviada en cada petición del servidor, a diferencia de las cookies .
c) Utilizan un número mínimo de peticiones al servidor para reducir el tráfico de la red.
d) Previenen pérdidas de información cuando se desconecta de la red.
e) La información es guardada por domino web (incluye todas las páginas del dominio).

2.3. Métodos y propiedades

Ambos objetos de almacenaje proveen los mismos métodos y propiedades:

- setItem(clave, valor) – almacenar un par clave/valor.
- getItem(clave) – obtener el valor por medio de la clave.
- removeItem(clave) – eliminar la clave y su valor.
- clear() – borrar todo.
- key(índice) – obtener la clave de una posición dada.
- length – el número de ítems almacenados.

2.4. Ejemplos

2.4.1. Local Storage

Primero ingresamos los datos y los guardamos. Después, cerraremos y abriremos nuevamente la pestaña o ventana nuestro navegador, daremos clic en “cargar elementos” y nos daremos cuenta de que los datos que almacenamos inicialmente siguen ahí, gracias a la propiedad de LocalStorage.

```
<html>
  <head>
    <title>Ejemplo LocalStorage</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script>
 /*Funcion de Capturar, Almacenar datos y Limpiar campos*/
$(document).ready(function(){
  $('#boton-guardar').click(function(){
    /*Captura de datos escrito en los inputs*/
    var nom = document.getElementById("nombretxt").value;
    var apel = document.getElementById("apellidotxt").value;
    /*Guardando los datos en el LocalStorage*/
    localStorage.setItem("Nombre", nom);
    localStorage.setItem("Apellido", apel);
    /*Limpiando los campos o inputs*/
```

```
        document.getElementById("nombretxt").value = "";
        document.getElementById("apellidotxt").value = "";
    });
});

/*Funcion Cargar y Mostrar datos*/
$(document).ready(function(){
    $('#boton-cargar').click(function(){
        /*Obtener datos almacenados*/
        var nombre = localStorage.getItem("Nombre");
        var apellido = localStorage.getItem("Apellido");
        /*Mostrar datos almacenados*/
        document.getElementById("nombre").innerHTML = nombre;
        document.getElementById("apellido").innerHTML = apellido;
    });
});

</script>
</head>

<center><h1>Ejemplo - localStorage</h1>

<input type="text" placeholder="Nombre" id="nombretxt"><br><br>
<input type="text" placeholder="Apellido" id="apellidotxt"><br><br>
<button id="boton-guardar">Guardar</button><br>

<hr />
Nombre almacenado:
<label type="text" id="nombre"></label><br>
Apellido almacenado:
<label "text" id="apellido"></label><br>

<button id="boton-cargar">
Cargar elementos
</button>
</center>

<hr />

</body>
</html>
```

2.4.2. Session Storage

Los datos se almacenan en una sesión. De esta forma, cuando la pestaña o ventana del navegador se cierra, los datos se eliminan de forma permanente y no podrán cargarse nuevamente. Este método es usado para realizar aplicaciones más seguras y eliminar un almacenamiento innecesario de datos.

```
<html>
   <head>
      <title>Ejemplo sessionStorage</title>
     <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"</script>
      <script>
 /*Funcion de Capturar, Almacenar datos y Limpiar campos*/
$(document).ready(function(){
   $('#boton-guardar').click(function(){
      /*Captura de datos escrito en los inputs*/
      var nom = document.getElementById("nombretxt").value;
      var apel = document.getElementById("apellidotxt").value;
      /*Guardando los datos en el LocalStorage*/
      sessionStorage.setItem("Nombre", nom);
      sessionStorage.setItem("Apellido", apel);
      /*Limpiando los campos o inputs*/
      document.getElementById("nombretxt").value = "";
      document.getElementById("apellidotxt").value = "";
   });
});

/*Funcion Cargar y Mostrar datos*/
$(document).ready(function(){
   $('#boton-cargar').click(function(){
      /*Obtener datos almacenados*/
      var nombre = sessionStorage.getItem("Nombre");
      var apellido = sessionStorage.getItem("Apellido");
      /*Mostrar datos almacenados*/
      document.getElementById("nombre").innerHTML = nombre;
      document.getElementById("apellido").innerHTML = apellido;
   });
});

</script>
</head>
```

```
<body>
 <center><h1>Ejemplo - sessionStorage</h1>
  <input type="text" placeholder="Nombre" id="nombretxt"><br><br>
  <input type="text" placeholder="Apellido" id="apellidotxt"><br><br>
  <button id="boton-guardar">Guardar</button><br>
  <hr />
  Nombre almacenado:<label type="text" id="nombre"></label><br>
  Apellido almacenado:<label "text" id="apellido"></label><br>
  <button id="boton-cargar">Cargar elementos</button>
 </center>
 <hr />
</body>
</html>
```

3.- INDEXEDDB

IndexedDB es un API de bajo nivel para poder almacenar datos estructurados en la parte del cliente. Estos datos pueden ser también ficheros o blobs. **IndexedDB** utiliza índices para poder realizar búsquedas de alto rendimiento.
Aunque existen otros mecanismos de almacenamiento en el cliente, como pueden ser las **Cookies** o la **API WebStorage** estos presentan unas limitaciones de tamaño, unos *4kb* en las **Cookies** y hasta los *10Mb* en la **API WebStorage**. Sin embargo la base de datos **IndexedDB** nos permite almacenar cantidades mayores de datos, llegando en algunas implementaciones a los *250Mb*.

Las principales características que definen a IndexedDB son:

a) **Almacena pares clave/valor**: IndexedDB es un almacén de objetos, es decir que podemos almacenar cualquier tipo de objeto dentro de la base de datos. Si bien cada objeto tiene asociada una clave que lo identificará de una forma única. A dicha clave va asociado el valor que es el objeto en sí.

b) **Es Asíncrona**: Para no penalizar el rendimiento del cliente, la base de datos IndexedDB funciona de forma asíncrona. Esto nos permite almacenar grandes cantidades de datos sin que estemos penalizando a la respuesta de renderizado del navegador. Cuando veamos el API de IndexedDB veremos que las respuestas de sus métodos son asíncronas. Si bien podemos encontrar algunas implementaciones de IndexedDB utilizando promesas.

c) **Soporta Transacciones**: Las operaciones que realicemos sobre IndexedDB se realizan mediante transacciones (incluidas las lecturas). Es decir, podremos deshacer ciertas operaciones en el caso de que se produzca un fallo dentro de la transacción.

d) **Restricción de Dominio**: Una base de datos IndexedDB solo pertenece a un dominio, es por ello que solo podremos acceder al contenido de la información que alberga cuando lo operemos desde el dominio al que corresponde. De esta manera mantenemos la seguridad de los datos almacenados.

e) **Gran Capacidad de Almacenamiento**: Ya hemos visto que IndexedDB viene a paliar los límites de almacenamiento que aparecen en las Cookies o en la API WebStorage, llegando a poder almacenar hasta *250Mb*.

f) **Soporte Almacenamiento Binario:** Dentro de IndexedDB podemos almacenar contenido binario como un ArrayBuffer de objetos u objetos Blob.

3.1. Conceptos Básicos

3.1.1. Base de Datos: IDBDatabase

La base de datos se representa mediante un objeto IDBDatabase. Por cada dominio podremos crear tantas bases de datos como queramos.

La base de datos se identifica por un *nombre de base de datos* y una *versión*. Solo puede existir una única *versión* a la vez. Es por ello que si queremos cambiar de *versión* deberemos de realizar una migración de los datos de la versión actual.

3.1.2. Almacén de Objetos: IDBObjectStore

El almacén de objetos, representado por el objeto **IDBObjectStore** es el conjunto de datos relativos a un concepto. Sería similar a una tabla dentro de una base de datos relacional.
Dentro del almacén de objetos encontraremos los datos. Cada uno de los datos es un registro. Lo que caracteriza a cada registro es que tiene una clave asociada.

3.1.3. Índices: IDBIndex

Los índices nos permiten optimizar las búsquedas dentro de **IndexedDB**. La base de datos se mantendrá ordenada por los índices que definamos. El objeto IDBIndex nos ayuda a definir los índices.
Es importante definir los índices mediante IDBIndex ya que será necesario para poder realizar búsquedas con filtros.

3.1.4. Transacciones: IDBTransaction

Como hemos comentado el acceso de lectura y escritura sobre la base de datos se hace mediante transacciones. El objeto que representa una transacción es IDBTransaction. La transacción nos ofrecerá métodos error, abort y complete para poder gestionar el resultado de la transacción.

3.1.5. Consultas: IDBRequest

Las consultas que realicemos sobre la base de datos **IndexedDB** las vamos a gestionar mediante un objeto IDBRequest. Tendremos un evento onsuccess que nos avisará cuando la consulta haya sido satisfactoria.

3.1.6. Punteros o Cursores: IDBCursor

En el caso de que la consulta sobre la base de datos **IndexedDB** devuelva un conjunto múltiples de datos deberemos de manejar un cursor (o puntero) para poder recorrer el conjunto de objetos devueltos como respuesta. El objeto que nos ayuda a manejar los cursores es IDBCursor.

3.1.7. Rango de Clave: IDBKeyRange

Si queremos gestionar un subconjuto de los elementos almacenados deberemos de apoyarnos en los índices, en concreto en el elemento IDBKeyRange. El elemento IDBKeyRange nos ayuda a establecer filtros sobre los datos para que el contenido devuelto solo corresponda a un rango.