1 Almacenamiento local en JavaScript

...

1.1 Sumario

1.2 Cookies

Las cookies, de nombre más exacto **HTTP cookies**, es una tecnología que en su día inventó el navegador Netscape, actualmente definida en el estándar RFC 6265, y que consiste básicamente en información enviada o recibida en las cabeceras HTTP y que queda almacenada localmente en el lado del cliente durante un tiempo determinado. En otras palabras, es información que queda almacenada en el dispositivo del usuario y que se envía hacia y desde el servidor web en las cabeceras HTTP.

Cuando un usuario solicita una página web, el servidor envía el documento, cierra la conexión y se olvida del usuario. Si el mismo usuario vuelve a solicitar la misma u otra página al servidor, será tratado como si fuera la primera solicitud que realiza. Esta situación puede suponer un problema en muchas situaciones y las *cookies* son una técnica que permite solucionarlo.

Con las cookies, el servidor puede enviar información al usuario en las cabeceras HTTP de respuesta y esta información queda almacenada en el dispositivo del usuario. En la siguiente solicitud que realice el usuario la *cookie* es enviada de vuelta al servidor en las cabeceras HTTP de solicitud. En el servidor podemos leer esta información y así "recordar" al usuario e información asociada a él.

Recuerda: Para los ejemplos que aquí aparecen. Si ejecutas un archivo .html 'en local' (no desde un servidor), las cookies no te funcionarán en el Chrome, utiliza el Firefox para probar los códigos.

1.2.1 Crear cookies

Una *cookie* es un *string* (cadena de texto) que contiene parejas **parametro=valor** separadas por **;** de la siguiente forma:

<nombre>=<valor>; expires=<fecha>; max-age=<segundos>; path=<ruta>; domain=<dominio>; secure; httponly;

Los parámetros son:

- **<nombre>=<valor>**

  Requerido. **<nombre>** es el nombre (*key*) que identifica la *cookie* y **<valor>** es su valor. A diferencia de las *cookies* en PHP, en JavaScript se puede crear una *cookie* con un valor vacío (**<nombre>=**).

- **expires=<fecha>** y **max-age=<segundos>**

  Opcional. Ambos parámetros especifican el tiempo de validez de la cookie. expires establece una fecha (ha de estar en formato UTC) mientras que max-age establece una duración máxima en segundos. max-age toma preferencia sobre expires. Si no se especifica ninguno de los dos se creará una session cookie. Si es max-age=0 o expires=fecha-pasada la cookie se elimina.

- **path=<ruta>**

  Opcional. Establece la ruta para la cuál la cookie es válida. Si no se especifica ningún valor, la cookie será válida para la ruta la página actual.

- **domain=<dominio>**

Opcional. Dentro del dominio actual, subdominio para el que la cookie es válida. El valor predeterminado es el subdominio actual. Establecer domain=.miweb.com para una cookie que sea válida para cualqueir subdominio (nota el punto delante del nombre del dominio). Por motivos de seguridad, los navegadores no permiten crear cookies para dominios diferentes al que crea la cookie (same-origin policy).

- **secure**

  Opcional. Parámetro sin valor. Si está presente la cookie sólo es válida para conexiones encriptadas (por ejemplo mediante protocolo HTTPS).

- **HttpOnly**

  Opcional. Parámetro no disponible en JavaScript ya que crea cookies válidas sólo para protocolo HTTP/HTTPS y no para otras APIs, incluyendo JavaScript.

Aquí puedes ver el resto de parámetros.

1.2.2 La propiedad *document.cookie*

La propiedad *document.cookie* es todo lo que se necesita para trabajar con *cookies* del lado del cliente con JavaScript. A través de ella podemos crear, leer, modificar y eliminar *cookies*. Veamos cada uno de los casos.

- **Crear *cookies***

  Establecer una *cookie* en JavaScript es tan fácil como crear el *string* que define la *cookie* y asignarlo a *document.cookie*. Por ejemplo:

```
document.cookie = "nombrecookie=valorcookie; max-age=3600; path=/";
```

  Si queremos crear varias *cookies*, tenemos que hacer este paso una vez para cada una. Por ejemplo, con el siguiente código se crearían las *cookies* **comida_favorita** y **color_favorito**:

```
document.cookie = "comida_favorita=arroz; max-age=3600; path=/";
document.cookie = "color_favorito=amarillo";
```

  Este comportamiento se debe a que **document.cookie** no es un dato con un valor (*data property*), sino una propiedad de acceso con métodos **set** y **get** nativos (*accessor property*). Cada vez que se le asigna una nueva *cookie*, no se sobrescriben las *cookies* anteriores sino que la nueva se añade a la colección de *cookies* del documento.

  Recuerda que las *cookies* se envían en las cabeceras HTTP y, por tanto, deben estar correctamente codificadas. Puedes utilizar **encodeURIComponent()**:

```
var testvalue = "Hola mundo!";
document.cookie = "testcookie=" + encodeURIComponent( testvalue );
 //testcookie=Hola%20mundo!
```

  Si vas a utilizar el parámetro **expires**, recuerda que ha de ser una fecha en formato **UTC**. Te puede ser de ayuda el método Date.toUTCString(). Por ejemplo, una *cookie* con caducidad para el 1 de Febrero del año 2068 a las 11:20:

```
var expiresdate = new Date(2068, 1, 02, 11, 20);
var cookievalue = "Hola Mundo!";
document.cookie = "testcookie=" + encodeURIComponent( cookievalue ) + "; expires=" + expiresdate.toUTCString();
```

  Un modo alternativo de indicar la duración de la *cookie* es utilizando el parámetro **max-age=duración-máxima-en-segundos**, por ejemplo: 60*60*24*365 para un año.

  Por último, en relación a la duración de las **cookies** antes de que el navegador las borre, debes saber que si no se específica fecha de caducidad o no se especifica su duración la *cookie* será válida sólo para la sesión actual.

- **Modificar *cookies***

  Como vimos ahora, cada vez que se asigna una *cookie* a *document.cookie*, esta es añadida a la colección de *cookies* del documento. Esto es verdad excepto si la *cookie* asignada tiene un identificador que ya existe. En este caso se modifica la *cookie* existente en lugar de añadir una más.

  Por ejemplo, podemos crear la siguiente *cookie* con identificador nombre y valor Miguel:

```
document.cookie = "nombre=Miguel";
```

Si queremos modificar el valor, por ejemplo cambiarlo por Juan:

```
document.cookie = "nombre=Juan";
```

Es importante tener en cuenta que si una *cookie* se crea para un dominio o para un *path* determinado y se quiere modificar, el dominio y el *path* han de coincidir. De lo contrario se crearán dos *cookies* diferentes válidas para cada *path* y dominio. Por ejemplo, imaginemos que estamos en «miweb.com/blog» (el valor predeterminado del *path* es en este caso /**blog**):

```
// Supongamos que estamos en "miweb.com/blog"
// y creamos las siguientes cookies

// Creamos la cookie para el path "/"
document.cookie = "nombre=Miguel; path=/";

// Con la siguiente linea se crea una nueva cookie para el path "/blog" (valor por defecto)
// pero no se modifica la cookie "nombre" anterior porque era para un path diferente
document.cookie = "nombre=Juan";

// Con la siguiente línea SI se modifica la cookie "nombre" del path "/" correctamente
document.cookie = "nombre=Juan; path=/";
```

- **Eliminar *cookies***

Para eliminar una *cookie* desde JavaScript se debe asignar una fecha de caducidad (**expires**) pasada o un **max-age** igual a cero. En ambos casos da igual el valor que se le asigne a la *cookie* porque se eliminará pero ha de darse el nombre de la *cookie* aunque sea sin valor.

Por ejemplo, creamos la *cookie* con el identificador **nombre** y valor **Miguel** igual que antes:

```
document.cookie = "nombre=Miguel";
```

Si queremos eliminarla:

```
document.cookie = "nombre=; expires=Thu, 01 Jan 1970 00:00:00 UTC";

// O con max-age
document.cookie = "nombre=; max-age=0";
```

Al igual que ocurría con la modificación de *cookies*, para la eliminación el **path** y el **domain** también tienen que coincidir:

```
// Se crean dos cookies con el mismo identificador
// para dos paths diferentes
document.cookie = "nombre=Miguel; path=/noticias";
document.cookie = "nombre=Juan; path=/blog";

// Solo se elimina la cookie del path /noticias
document.cookie = "nombre=; max-age=0; path=/noticias";
```

- **Leer y obtener el valor de las *cookies***

Puede que obtener el valor sea el paso más tedioso de trabajar con *cookies* en JavaScript, y es que no hay un método de lectura directo para cada *cookie* individual. Sólo se puede obtener un *string* con todas las *cookies* válidas para el documento y manipular el *string* hasta encontrar el nombre y valor de la *cookie* que queremos.

El *string* con todas las *cookies* se obtiene del siguiente modo:

```
var lasCookies = document.cookie;
```

Y tiene el siguiente formato:
"cookie1=valor1;cookie2=valor2;cookie3=valor3[;...]"

Fijarse en que:

- ◊ El *string* sólo contiene pares de nombre de la *cookie* y su valor. No se puede acceder a otros parámetros a través de *document.cookie*.
- ◊ Sólo se obtienen las *cookies* válidas para el documento actual. Esto implica que *cookies* para otros *paths*, dominios o *cookies* caducadas no se pueden leer. Aunque en una página puedan crearse *cookies* para otros subdominios y *paths*, sólo se pueden leer las que san válidas para el subdominio y *path* actual.

1.2.3 Ejemplo sencillo de creación y lectura de *Cookies*

En este ejemplo tenemos un botón y una etiqueta. Se crea una cookie que guarda las veces que pulsamos el botón en cada sesión del navegador. El numero de pulsaciones se mostrará en un elemento **label**.

- **HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prueba cookie</title>
</head>
<body>


    <input type="button" id="boton" value="Pulsa">
    <label id="labelContador">0</label>

    <script src="script.js"></script>
</body>
</html>
```

- **JavaScript**

```
//Iniciamos la cookie para esta sesión
document.cookie  = "pulsados=0; SameSite=None; Secure;";
 //console.log(document.cookie);

function alertCookie() {
    //Creamos un diccionario con el contenido de la Cookie
    let dCookie = {};
    const laCookie = document.cookie; // La cookie en formato string
    const aCookie = laCookie.split(";"); // Array de pares clave=valor existentes en la cookie
    console.log(aCookie);
for (let i = 0; i < aCookie.length; i++) { // Llenamos el diccionario con los pares clave:valor
        dCookie[aCookie[i].split('=')[0].trim()] = aCookie[i].split('=')[1].trim();
    }
     //console.log('El diccionario: ');
     //console.log(dCookie);

    //Cambiamos contenido de la Cookie
    dCookie['pulsados'] = parseInt(dCookie['pulsados']) + 1;
    textoCookie = 'pulsados=' + dCookie['pulsados'] + '; SameSite=None; Secure;';
    document.cookie = textoCookie;
     // console.log(document.cookie);

    //Mostramos en la 'label' el número de veces que se pulsó el botón al recargar la web
    document.getElementById('labelContador').innerText = dCookie['pulsados'];
}

//Añadimos un escuchador al evento click para el botón de la web
document.getElementById('boton').addEventListener('click',alertCookie,false);
```

1.2.4 Enlaces externos interesantes sobre *cookies*

- Cookies en developer Mozilla
- Introducción a las cookies
- Funciones de manejo de cookies
- Guía sobre uso de cookies AEPD

1.3 LocalStorage

El mecanismo de almacenamiento DOM es el medio a través del cual pares de clave/valor pueden ser almacenadas de forma segura para ser

recuperadas y utilizadas más adelante. La meta de este añadido es suministrar un método exhaustivo a través del cual puedan construirse aplicaciones interactivas (incluyendo características avanzadas tales como ser capaces de trabajar "sin conexión" durante largos períodos de tiempo).

El almacenamiento DOM está diseñado para facilitar una forma amplia, segura y sencilla para almacenar información alternativa a las *cookies*.

LocalStorage y **sessionStorage** son propiedades de HTML5, 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:

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

Los datos de almacenamiento DOM se eliminan junto con la memoria caché sin conexión cuando se hace clic en el botón Eliminar.

1.3.1 Ejemplo LocalStorage

En el siguiente ejemplo se muestra como se pueden almacenar en local dos **clave=valor** y luego leerlas. 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**

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>Ejemplo LocalStorage</title>
        <script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossor


</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="nombrelabel"></label><br>
Apellido almacenado:
<label "text" id="apellidolabel"></label><br>

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

<hr />

<script src="script.js"></script>


</body>
```

```
</html>
```

- **JavaScript**

```
function guardar() {
    // Leemos los datos escritos en los inputs
    const nom = document.getElementById('nombretxt').value;
    const apel = document.getElementById('apellidotxt').value;

    // Guardamos los datos en el LocalStorage
    localStorage.setItem('Nombre', nom);
    localStorage.setItem('Apellido', apel);

    // Limpiamos los inputs una vez leídos
    document.getElementById('nombretxt').value = "";
    document.getElementById('apellidotxt').value = "";
}

function cargar() {
    // Obtener los datos almacenados
    const nom = localStorage.getItem('Nombre');
    const ape = localStorage.getItem('Apellido');

    // Mostramos los datos almacenados
    document.getElementById('nombrelabel').innerText = nom;
    document.getElementById('apellidolabel').innerText = ape;

}


document.getElementById('boton-guardar').addEventListener('click', guardar, false);
document.getElementById('boton-cargar').addEventListener('click', cargar, false);
```

Nota: Si queremos guardar un un array o un JSON, recuerda que localStorage solo trabaja con texto, así:

```
// Creamos un array
var arrayNombres = ['ana', 'manuel', 'patricia', 'daniel'];
// Guardamos el array pero en formato texto
localStorage.setItem("nombres", JSON.stringify(arrayNombres));
//...
// Ya podemos leer los datos guardados en local
var arrayNGuardados = JSON.parse(localStorage.getItem("nombres"));
```

1.3.2 sessionStorage

Este es un objeto global (**sessionStorage**) que mantiene un área de almacenamiento que está disponible durante la sesión de página. Una sesión de página existe mientras el navegador esté abierto y sobrevive a recargas o restauraciones de páginas. Abrir una página en una nueva pestaña o en una ventana provoca que se cree una nueva sesión.

```
// Guardar datos en el almacén de la sesión actual
sessionStorage.setItem("username", "John");

// Acceder a algunos datos guardados
console.log( "username = " + sessionStorage.getItem("username"));
```

El objeto **sessionStorage** es más usado para manejar datos temporales que deberían ser guardados y recuperados si el navegador es recargado accidentalmente.

Para ver el funcionamiento se puede emplear el ejemplo anterior simplemente cambiando **localStorage** por **sessionStorage**.

Ejemplo con **sessionStorage**:

- **HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Local Storage</title>
</head>
<body>
    <h1>Pruebas con Local Storage</h1>

    <form action="#" id="formulario">
        <label for="clave">Clave : </label><input type="text" name="clave" id="clave"><br>
        <label for="valor">Valor : </label><input type="text" name="valor" id="valor"><br>
        <input type="submit" value="Guardar" id="guardar">
    </form>
    <div id="info">
        <select name="claves" id="claves">
            <option value="porDefecto">Selecciona clave</option>
        </select>
        <p id='infoValor'></p>
    </div>

    <script src="scripts.js"></script>
</body>
</html>
```

- **JavaScript**

```
const formulario = document.getElementById('formulario');
const claves = document.getElementById('claves');
const infoValor = document.getElementById('infoValor');

formulario.addEventListener('submit', e => {
    e.preventDefault();

    const clave = formulario.clave.value;
    const valor = formulario.valor.value;

    sessionStorage.setItem(clave, valor);
    claves.innerHTML += `<option>${clave}</option>`;

    formulario.reset();

    claves.addEventListener('change', () => {
        const contenido = sessionStorage.getItem(claves[claves.selectedIndex].textContent);
        console.log(contenido);
        infoValor.textContent = contenido;

    })

});
```

Volver