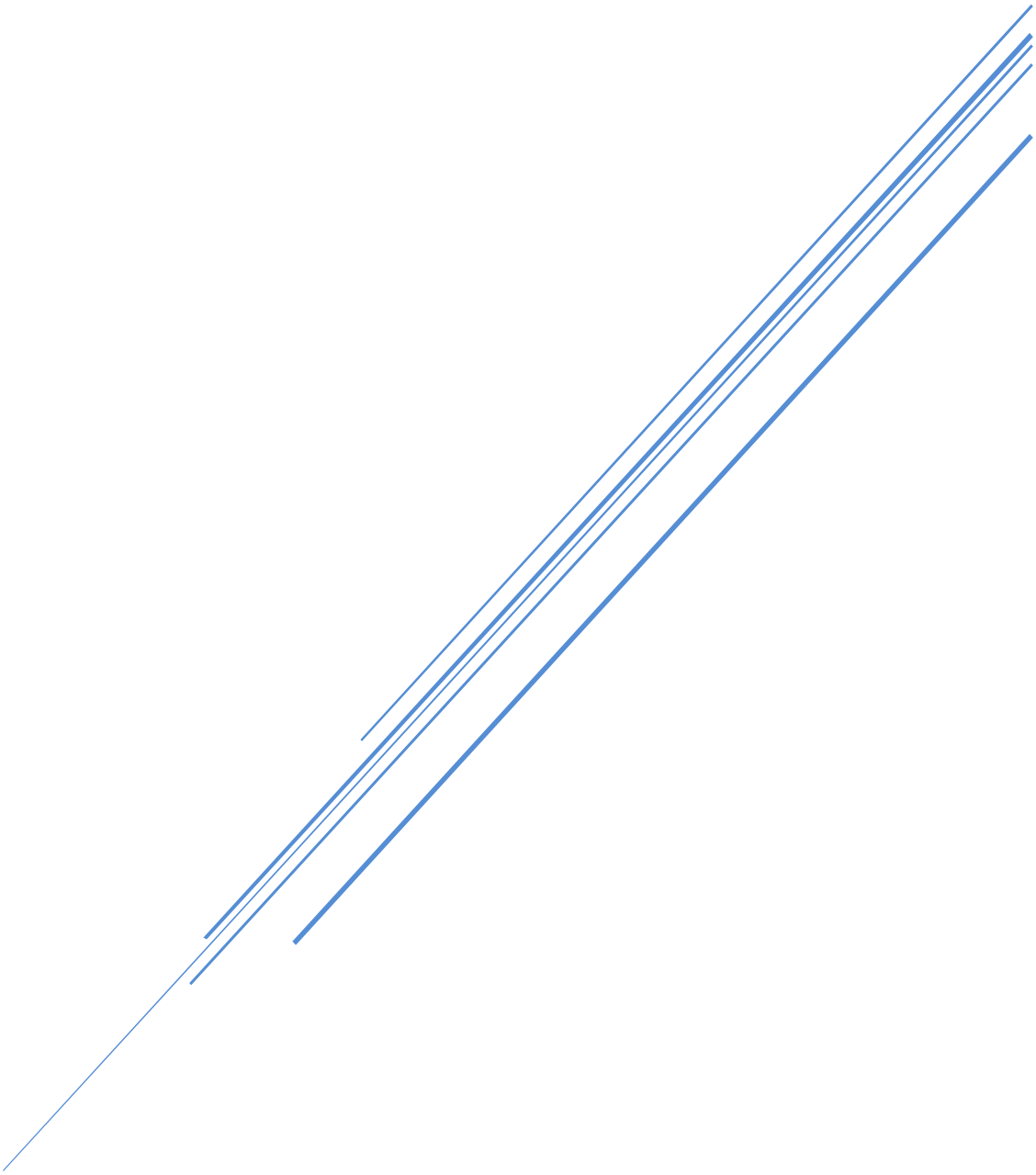
COOKIES

Utilidad, métodos y propiedades



Mario Osuna Ordoñez
Antonio Quesada Cuadrado

ÍNDICE

¿Qué son las HTTP cookies y cómo funcionan?

¿Qué son?

Tecnología inventada por el navegador Netscape, definida actualmente en el estándar RFC 6265.

Consiste en **información enviada** o recibida en las **cabeceras HTTP** y que queda almacenada localmente durante un tiempo determinado.

¿Por qué utilizarlas?

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.

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.

Se utilizan principalmente con tres propósitos:

- Gestión de Sesiones; como inicios de sesión, puntajes de juegos…
- Personalización; preferencias de usuario, temas y otras configuraciones.
- Rastreo; guardar y analizar el comportamiento del usuario.

¿Cómo funcionan?

Las cookies pueden ser persistentes o no persistentes. Son **persistentes,** si tienen un tiempo de expiración y si no lo tienen, se borran en el momento en que se cierra el navegador.

Las cookies **no persistentes** se usan para mantener abierta la sesión del usuario, ya que el servidor sabe por el identificador de la cookie a qué usuario pertenece cada sesión abierta en memoria.
Para evitar que cualquier página pueda recuperar el estado que tenía el usuario en otra web distinta, las cookies llevan asociadas siempre un dominio y sólo pueden crearse, modificarse o borrarse si pertenecen al mismo dominio o al subdominio del nivel superior**.**

Propiedad document.cookie

Esta propiedad de JavaScript nos permite guardar las cookies y volver a leerlas.

Para guardar una cookie escribiremos en principio:

document.cookie = "nombre=valor"

Cada cookie tiene un nombre, que es por lo que la identificaremos, y un valor que es el que puede cambiar dentro de la misma cookie, por ejemplo, si escribimos:

document.cookie = "color=azul";

Aquí se está guardando una cookie cuyo identificador es color y su valor es azul.

Para leer una cookie tan solo se pondrá document.cookie y para borrar una cookie concreta basta con reescribirla poniéndole una fecha de caducidad anterior a la actual.

Aparte de la clave y su valor, las cookies pueden contener atributos opcionales que se escriben después de la clave-valor, especificando la cookie que se va a crear o actualizar y precedido de un punto y coma.

path=path (p. ej.: '/'. '/midir'). Si no se especifica, por defecto corresponde a la ruta del documento actual. La ruta debe ser absoluta.

domain=domain (p. ej. 'example.com', 'subdomain.example.com'). Si no se especifica, su valor por defecto es de la ubicación actual del archivo(dirección web donde se aloja

max-age=duración-máxima-en-segundos Por ejemplo: 60*60*24*365 para un año.

expires=fecha-en-formato-GMTString Si no se especifica max-age ni expires, la cookie expirará al terminar la sesión actual.

secure La cookie sólo será transmitida en un protocolo seguro (HTTPS, SSL).

samesite Este atributo impide al navegador enviar esta cookie a través de peticiones de origen cruzado (donde el sitio está definido por el dominio registrable), lo que brinda cierta protección contra ataques de falsificación de solicitudes entre sitios. Los valores posibles son lax o strict o none

El valor strict impide que la cookie sea enviada por el navegador al sitio destino en contexto de navegador cross-site, incluso cuando sigue un enlace regular.

El valor lax sólo envía cookies a las peticiones de GET de ALTO NIVEL. Es suficiente para seguir al usuario, pero evitará muchos ataques CSRF.

El none valor indica explícitamente que no se aplicarán restricciones.

El valor de la cookie puede ser evaluado mediante encodeURIComponent() para asegurarse de que dicha cadena no incluya comas, punto y coma, ni espacios en blanco (lo cual no está permitido en el valor de una cookie).

Algunas implementaciones de agente de usuario soportan los siguientes prefijos de cookie:

__**Secure**- Señales para el navegador que solo deben incluirse en las perticiones de cookie transmitidas por un canal seguro.

__**Host**- Señales del navegador que además de la restricción de uso de cookies que provienen de un origen serugo, el ámbito de la cookie está limitado a un atributo path que proporciona el servidor.

Ejemplos:

```
//Creamos una cookie
let fecha = new Date();
alert(document.cookie = "expires="+ fecha);
alert(document.cookie = "username=user1; max-age=3600");
```

```
document.addEventListener("DOMContentLoaded", function () {
document.addEventListener("click",alertCookie); //Evento
    document.cookie = "nombre=antonio"; //Creación
    document.cookie = "apellidos=quesada cuadrado";
    //document.cookie = "apellidos = qc"; //Modificamos valor cookie
    function alertCookie() {
        alert(document.cookie);

    }

})
```

Propiedad Window.localStorage & Window.sessionStorage

Antes de HTML5, la información de las aplicaciones sólo se podía guardar en cookies.

Como una alternativa al document.cookie encontramos la interfaz Storage de la API de almacenamiento web, que provee acceso al almacenamiento de la sesión o al almacenamiento local para un dominio en particular, permitiendo añadir, modificar o eliminar elementos de datos almacenados.

Para trabajar con este almacenamiento se debe utilizar según se quiera los métodos Window.localStorage y Window.sessionStorage.

La diferencia más importante entre localStorage y sessionStorage, es que en localStorage se guarda la información de forma persistente en el navegador, es decir, si el usuario cierra el navegador y vuelve a entrar en la web, la información aun seguirá disponible. En cambio, en sessionStorage, la información sólo dura lo que dure la sesión del navegador. Al cerrar la ventana se perderá la información.

Window.localStorage

Los datos se encuentran almacenados localmente dentro del navegador del usuario. Pudiendo tener grandes cantidades de datos de forma local, sin afectar el rendimiento del sitio web. Los almacena sin fecha de vencimiento y se borra solo a través de JavaScript, o borrando la caché del navegador. Las claves y los valores son siempre cadenas de texto

El siguiente código accede al objeto local Storage actual y agrega un ítem al mismo usando Storage.setItem().

localStorage.setItem('miGato', 'Juan');

Para leer el ítem almacenado en localStorage es la siguiente

var cat = localStorage.getItem('miGato');

Para eliminar el ítem:

localStorage.removeItem('miGato');

Si se quieren eliminar todos los ítem se pondrá

localStorage.clear();

Para conocer la cantidad de ítems almacenados se puede utilizar : *localStorage.length*

Por ejemplo:

```
localStorage.setItem("Nombre", "Mario");
localStorage.setItem("Apellido", "Osuna");


let nombre = localStorage.getItem("Nombre");
let apellido = localStorage.getItem("Apellido");

console.log(nombre);
console.log(apellido);
```

```
document.addEventListener("DOMContentLoaded", function () {
  // Obtiene el campo de texto que vamos a moniterear
var field = document.getElementById("field");

// Espera por cambios en el campo de texto
field.addEventListener("change", function() {
  // Almacena el resultado en el objeto de almacenamiento de sesión
  localStorage.setItem("autosave", field.value);
});
});
```

Window.sessionStorage

Para emplear sessionStorage se utilizan los mismos métodos que con LocalStorage pero añadiendo como es lógico sessionStorage.

Por ejemplo: sessionStorage.setItem('myCat', 'Tom');

Dado que Sessionstorage no almacena siempre los datos sino solo durante la sesión que se esté navegando, se pueden realizar comprobaciones en el código si la página se ha recargado y así restaurar los datos sin perderlos

Por ejemplo;

```
document.addEventListener("DOMContentLoaded", function () {
  // Obtiene el campo de texto que vamos a moniterear
var field = document.getElementById("field");

// Verificamos si tenemos algún valor auto guardado
// (esto solo ocurrirá si la página es recargada accidentalmente)
if (sessionStorage.getItem("autosave")) {
  // Restaura el contenido al campo de texto
  field.value = sessionStorage.getItem("autosave");
}

// Espera por cambios en el campo de texto
field.addEventListener("change", function() {
  // Almacena el resultado en el objeto de almacenamiento de sesión
  sessionStorage.setItem("autosave", field.value);
});
});
```

Referencias

https://masqueweb.es/trabajar-con-cookies-js

https://www.humanlevel.com/articulos/desarrollo-web/que-son-y-como-funcionan-las-cookies.html

https://rolandocaldas.com/html5/localstorage-en-html5

https://developer.mozilla.org/es/docs/Web/API/Window/sessionStorage

https://stackoverflow.com/questions/3220660/local-storage-vs-cookies

https://aprende-web.net/javascript/js14_1.php

https://developer.mozilla.org/es/docs/Web/API/Window/localStorage

https://developer.mozilla.org/es/docs/DOM/document.cookie

http://www.webtutoriales.com/articulos/cookies-vs-localstorage-vs-sessionstorage

https://developer.mozilla.org/es/docs/Web/HTTP/Cookies

https://developer.mozilla.org/es/docs/Tools/Storage_Inspector

https://cybmeta.com/que-son-las-cookies-y-como-funcionan

https://cybmeta.com/cookies-en-javascript