JavaScript: Persistencia de datos en cliente

Alicia Vázquez
@aliciaFPInf

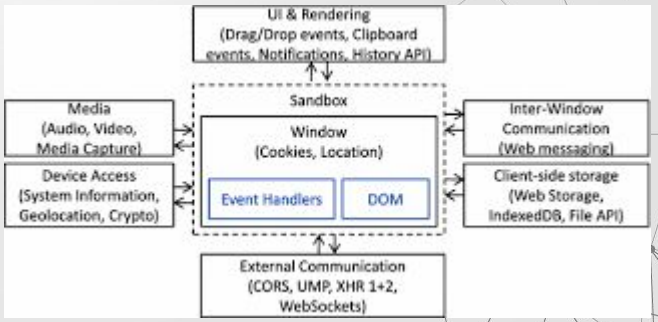
Persistencia

El protocolo **HTTP** es **stateless** (sin estado), es decir que se trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior, de modo que la comunicación se compone de pares independientes de solicitud y respuesta. **El servidor no mantiene memoria de nada de lo ocurrido anteriormente.**

Por otro lado, **Javascript** no tiene acceso directo ni al sistema de archivos del cliente ni del servidor. JS se ejecuta en un entorno llamado **sandbox** (entorno aislado).

Para poder guardar información en el cliente, los navegadores proporcionan librerías que forman parte de las últimas versiones de JS ES6 .

- Las principales son las **Cookies** y **LocalStorage** últimamente también Web SQL Databases pero no es soportado por todos los navegadores.



01

Cookies

enlace

Cookies

Las cookies son pequeños archivos de texto que se **almacenan en tu dispositivo** cuando visitas una página web. Estos archivos contienen información sobre tus interacciones con la página, y se utilizan para mejorar la experiencia de usuario y personalizar el contenido.

Son pequeños ficheros de texto en los que podemos guardar información. Es directamente el servidor quien pide al usuario permiso para poder almacenar información en local.

- Javascript tiene comandos específicos para manipular esos ficheros.
- Se guardan variables y su valor *(key, value)*. Máximo 4 KB por cookie, si nos pasamos se truncará.
- Las cookies sólo se pueden modificar desde el dominio que las ha creado. Máximo 20 cookies por dominio.
- Las cookies son las que, por ejemplo, mantienen una sesión (ej. Cesta compra) o ayudan a la publicidad contextual (ej. Hábitos de navegación).

Cookies: tipos

Hay varios tipos de cookies, como las **temporales** o de **sesión**, que se eliminan automáticamente cuando cierras tu navegador, y las **permanentes** o **persistentes**, que se almacenan en tu dispositivo por un período de tiempo determinado. Las cookies también pueden ser **propias**, creadas por la página web que visitas, o de terceros, creadas por otras empresas.

Las cookies tienen diferentes finalidades, como las **técnicas** o **necesarias**, que permiten el funcionamiento básico de la página web y no se pueden desactivar. También hay cookies de **análisis**, que recopilan información sobre la actividad del usuario en la página para mejorar la experiencia, y cookies **publicitarias**, que se utilizan para mostrar anuncios personalizados y relevantes.

Otro tipo de cookies son las de personalización, que almacenan tus **preferencias** y **configuraciones** en la página, y las de redes sociales, que permiten compartir contenido en redes sociales desde la página web.

Cookies: Configuración

Las páginas web tienen la obligación de mostrar una opción para configurar las cookies, donde se pueden desactivar algunos tipos de cookies.

Si se borran las cookies, se perderán los inicios de sesión y las preferencias, y se tendrá que volver a iniciar sesión y reconfigurar las cookies.

Si se desactivan las cookies de análisis, no se registrará la actividad del usuario en la web, y si se desactivan las cookies de preferencias, la información de configuración no se recordará. La privacidad aumentará al desactivar cookies, pero también se perderá la personalización en la experiencia de navegación.

Se recomienda encontrar un equilibrio entre la privacidad y la experiencia.

Cookies: Implementación

Cada cookie almacena los siguientes datos:

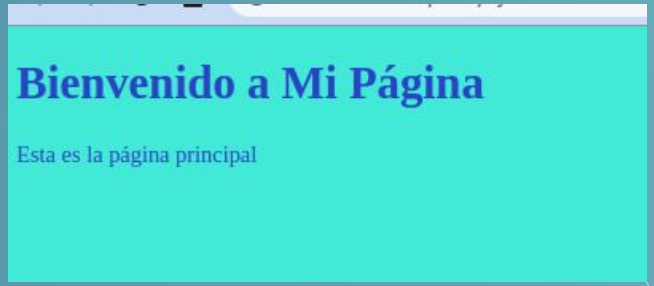
- **Nombre de la cookie** (obligatorio)
- **Valor** de la misma
- **expires**: timestamp en que se borrará (si no pone nada se borra al salir del dominio, al cerrar el navegador)
- **max-age**: en lugar de expires podemos indicar aquí los segundos que durará la cookie antes de expirar
- **path**: ruta desde dónde es accesible (/: todo el dominio, /xxx: esa carpeta y subcarpetas). Si no se pone nada sólo lo será desde la carpeta actual
- **domain**: dominio desde el que es accesible. Si no ponemos nada lo será desde este dominio y sus subdominios
- **secure**: si aparece indica que sólo se enviará esta cookie con https

Se puede acceder a las cookies desde **document.cookie** que es una cadena con las cookies de nuestras páginas. Para trabajar con ellas conviene que creemos funciones para guardar, leer o borrar cookies

```
function creaCookies() {
    //Creamos la cookie
    document.cookie = "nombre=Alicia";
    document.cookie = "comidaPreferida=Risotto";
}

function visualizaCookies() {
    //Visualizamos
    //console.log(document.cookie);

    //Todas las cookies estan sepradas por ; por lo que es interesante crear un array para
    var misCookies = document.cookie;
    var arrayCookies = misCookies.split(";");

    var comida;
    var nombre;
    arrayCookies.map(
        (cookie) => {
            c = cookie.split("=");
            console.log(c);
            if (c[0].includes("comidaPreferida")) {
                comida = c[1];
            }
            if (c[0] == "nombre") {
                nombre = c[1];
            }
        }
    );
    alert("A  " + nombre + " le encanta el" + comida);
}

function eliminarCookies() {
    document.cookie.split(";").forEach(function (c) {
        document.cookie = c + '=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
    });
```

Ejercicio 1: Cookies

Utiliza alguna de las páginas que ya has hecho (la de inmobiliaria o la de videojuegos) y detectando si es la primera vez que entras en el dominio te abra una ventana de configuración, donde mediante un pequeño formulario elijas el color de fondo y el color de la fuente. Lo guardes en una cookie y luego lo recuperes cada vez que entres.



Ejercicio 1: Cookies

Evento desde la página de popUp

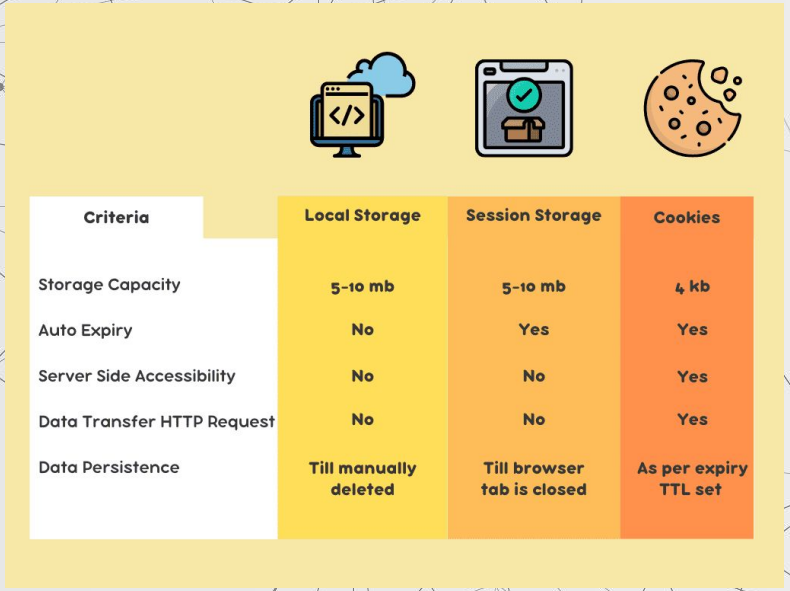
```
//Evento que se lanza al cerrar esta ventana y así avisamos al que nos ha abierto.
window.addEventListener("beforeunload", function () {
   window.opener.postMessage("ventana_cerrada", "*");
});
```

Evento que escuchamos para saber si nos han cerrado o no la ventana.

```
//Especificamos el evento que nos indica que la ventana ha sido cerrada
       window.addEventListener("message", function (event) {
           if (event.data === "ventana_cerrada") {
           // Aplicamos la configuración guardada en la cookie a la página
            ....
            }
       });
```

02

API Web Storage



Criteria	Local Storage	Session Storage	Cookies
Storage Capacity	5-10 mb	5-10 mb	4 kb
Auto Expiry	No	Yes	Yes
Server Side Accessibility	No	No	Yes
Data Transfer HTTP Request	No	No	Yes
Data Persistence	Till manually deleted	Till browser tab is closed	As per expiry TTL set

API Web Storage

Hemos visto las cookies como mecanismo de almacenamiento de información en el lado del cliente, sin embargo existen maneras más actuales de hacerlo,

API WebStorage nos provee una forma de guardar información del lado del cliente en forma de **Key/Valor**, y a diferencia de las cookies esta información sólo es persistida del lado del cliente y no será enviada en cada petición que realicemos.

WebStorage nos brinda un conjunto de atributos y métodos para poder manipularlos. Existen dos implementaciones para ello:

- **LocalStorage**, nos permite almacenar información (strings) de manera permanente para múltiples ventanas o pestañas, de esta manera si cerramos el navegador no perdemos la información guardada a menos que decidamos eliminarla manualmente. Podemos almacenar entre 5 y 10 mb de información y a diferencia de las cookies, no será enviada al servidor.
- **SessionStorage**, es igual que el localStorage, con la salvedad que si cerramos el navegador la información se pierde.

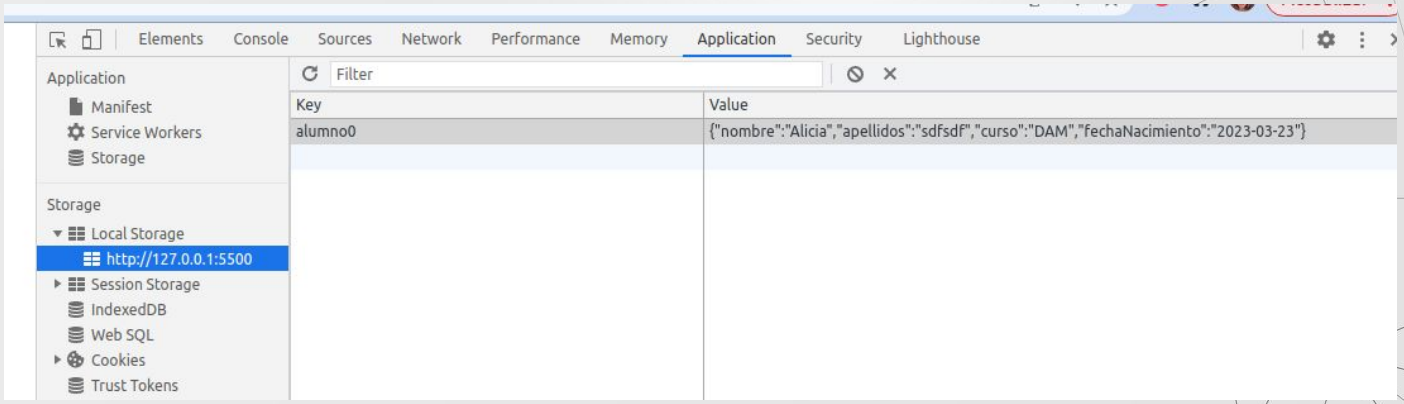
Nota: Si abrimos el devtools del navegador, en la pestaña "application" podemos ver la sección de Storage con todas las herramientas de persistencia que nos provee el navegador.

API Web Storage

Al contrario que las cookies, los objetos de almacenaje web **no se envían al servidor** en cada petición. Debido a esto, podemos almacenar mucha más información. La mayoría de los navegadores modernos permiten almacenar, como máximo, **5 megabytes de datos** y tienen opciones para configurar estos límites.

Además, a diferencia de las cookies el servidor no puede manipular los objetos de almacenaje vía cabeceras HTTP, todo se hace vía JavaScript.

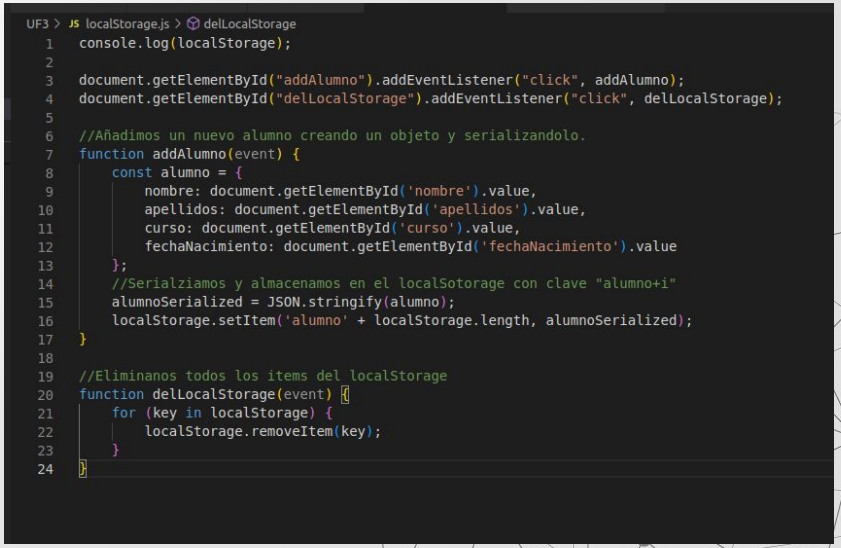
El almacenaje está vinculado al orígen (al triplete dominio/protocolo/puerto). Esto significa que distintos protocolos o subdominios tienen distintos objetos de almacenaje, no pueden acceder a otros datos que no sean los suyos.



API Web Storage

Tanto el **localStorage** como el **sessionStorage** tienen 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.



```
console.log(localStorage);
document.getElementById("addAlumno").addEventListener("click", addAlumno);
document.getElementById("delLocalStorage").addEventListener("click", delLocalStorage);

//Añadimos un nuevo alumno creando un objeto y serializandolo.
function addAlumno(event) {
    const alumno = {
        nombre: document.getElementById('nombre').value,
        apellidos: document.getElementById('apellidos').value,
        curso: document.getElementById('curso').value,
        fechaNacimiento: document.getElementById('fechaNacimiento').value
    };
    //Serialziamos y almacenamos en el localSotorage con clave "alumno+i"
    alumnoSerialized = JSON.stringify(alumno);
    localStorage.setItem('alumno' + localStorage.length, alumnoSerialized);
}
//Eliminanos todos los items del localStorage
function delLocalStorage(event) {
    for (key in localStorage) {
        localStorage.removeItem(key);
    }
}
```

Ejercicio 2: API WebStorage

Del ejercicio anterior, cambia las cookies por el localStorage.