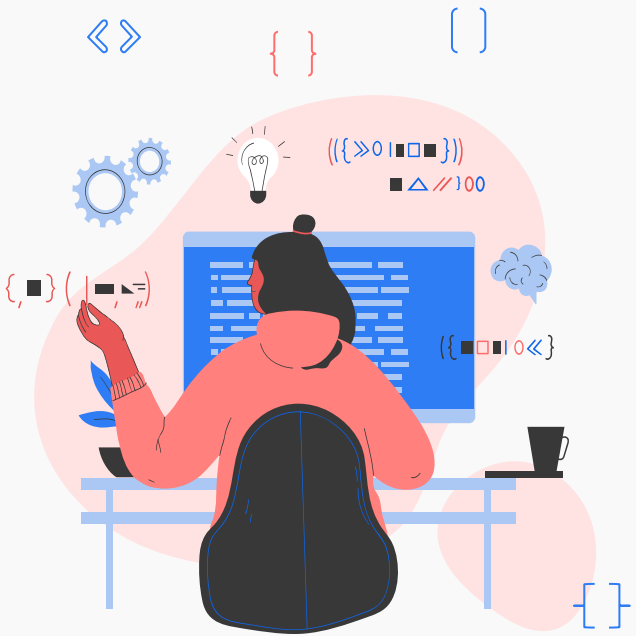
Desarrollo Web en Entorno Cliente

Tema 4 – Almacenamiento web en el lado cliente

Marina Hurtado Rosales
marina.hurtado@escuelaartegranada.com



Indice de contenidos

Fundamentos del almacenamiento web en cliente

Introducción

Hasta ahora hemos trabajado con JavaScript para interactuar con el usuario mediante el DOM, eventos y formularios.

Sin embargo, toda la información que manejábamos se perdía al recargar la página.

> El almacenamiento web permite **guardar datos en el navegador del usuario**, de forma que puedan recuperarse más adelante, incluso después de cerrar la página.

Esto nos permite crear aplicaciones web más completas, persistentes y cercanas al comportamiento de una aplicación de escritorio.

¿Qué es el almacenamiento web?

El almacenamiento web es una funcionalidad proporcionada por los navegadores modernos que permite a las aplicaciones web:

- Guardar información en el **navegador** del usuario (en el **lado cliente**)
- **Recuperar** esa información posteriormente.
- Evitar depender siempre de un servidor.

Ejemplos de uso:

- Guardar información sobre preferencias del usuario (idioma, tema...).
- Recordar datos de un formulario.
- Guardar datos de sesión.
- Guardar el estado de una aplicación, para que en caso de error se pueda seguir desde el punto en el que estaba el usuario en su última visita.

Fundamentos del almacenamiento

Para que una aplicación web pueda almacenar datos en el cliente, el navegador debe reservar espacio de memoria.

Características importantes:

- Los navegadores suelen permitir entre 5 y 10 MB de almacenamiento.
- Los datos se asocian a un **origen** (protocolo + dominio + puerto)
- Por defecto, el almacenamiento en cliente está **dividido por página**. Una página no puede acceder a los datos de otra página, salvo que provengan del mismo dominio.

Almacenamiento temporal y permanente

Se puede fijar el tiempo de vida que los datos pueden permanecer almacenados:

Almacenamiento temporal

- Los datos existen mientras la pestaña o ventana esté abierta.
- Al cerrar la pestaña, los datos se eliminan.

Almacenamiento permanente

- Los datos persisten un tiempo indeterminado, aunque se cierre el navegador.
- Estos datos sólo se eliminan si el usuario o la aplicación web los borra.
- Es importante seguir algunas medidas de seguridad, como la encriptación, en el caso de almacenar datos de forma permanente en el navegador.

Tipos de almacenamiento en cliente

Tipos de almacenamiento en cliente

Tipo	Uso principal
Almacenamiento web: localStorage	Datos persistentes
Almacenamiento web: sessionStorage	Datos de sesión
Cookies	Pequeños datos enviados al servidor en cada petición
IndexedDB	Bases de datos en cliente usadas para grandes volúmenes de datos

En este tema nos centraremos principalmente en el **almacenamiento web**, por se el más utilizado desde JavaScript.

Almacenamiento local y de sesión

Almacenamiento web: local y de sesión

El almacenamiento web usa la API de JavaScript que permite el almacenamiento local o a nivel de sesión.

Se realiza mediante dos objetos proporcionados por el objeto *window* (navegador):

- **localStorage**
- **sessionStorage**

Al igual que cualquier objeto de JavaScript, localStorage y sessionStorage funcionan usando pares **clave-valor.**

Sin embargo, aunque localStorage y sessionStorage se acceden como objetos de JavaScript, **disponen de una API propia** y un comportamiento diferente a los objetos normales:

- Sólo permiten almacenar **cadenas de texto**.
- Las propiedades del objeto son persistentes, es decir, si un usuario recarga la página, las propiedades del objeto siguen existiendo.

localStorage

Este objeto permite almacenar datos de **forma permanente** en el navegador.

Estos datos sólo se pueden borrar si lo hace la propia aplicación web o si el usuario realiza un **borrado manual** a través del propio navegador.

Los datos sólo pueden **compartirse** en aquellos sitios que provengan del **mismo dominio**.

Hay que considerar que estos datos dependen del navegador utilizado.

Usos habituales:

- Preferencias del usuario
- Configuración de la aplicación
- Datos que deben mantenerse entre sesiones

sessionStorage

Este objeto permite almacenar datos de **sólo durante la sesión actual**.

Estos datos tienen un tiempo de vida limitado que dependerá del tiempo que la ventana del navegador o pestaña esté abierta.

Al igual que con el localStorage, los datos sólo pueden **compartirse** en aquellos sitios que provengan del **mismo dominio**.

Sin embargo, cada pestaña tiene su propio almacenamiento y no se comparten datos entre pestañas.

Usos habituales:

- Estados temporales
- Contadores de sesión
- Datos intermedios de una aplicación

localStorage vs sessionStorage

localStorage	sessionStorage
Permanente	Temporal
Persiste al cerrar el navegador	Se borra al cerrar pestaña
Compartido por dominio	No compartido entre pestañas
Preferencias	Datos temporales

Funcionamiento del almacenamiento

El almacenamiento web funciona mediante pares **clave-valor**:

- La clave es una cadena de texto
- El valor se almacena siempre como texto

```
localStorage.setItem("nombre", "Ana");
```

Métodos de almacenamiento web

Al igual que sucede con otros objetos, con **localStorage** y **sessionStorage** se puede **mostrar todo el contenido** usando cualquier **sentencia iterativa**.

La API de almacenamiento de JavaScript nos proporciona además los siguientes métodos para estos objetos:

- **setItem(clave, valor)**: guarda un elemento con su clave asociada
- **getItem(clave)**: recupera un determinado elemento
- **removeItem(clave)**: permite borrar un elemento a partir de su clave
- **clear()**: Borra todo el contenido almacenado en el objeto

Es importante recordar que **los objetos de almacenamiento sólo permiten guardar contenido en formato cadena**. Si se quiere guardar otro tipo de datos, se debe realizar la conversión manualmente.

Ejemplo básico de uso

```
// Guardar datos en localStorage
localStorage.setItem("nombre", "Ana");
localStorage.setItem("edad", "25");

// Obtener un dato concreto
let nombre = localStorage.getItem("nombre");

// Recorrer y mostrar todos los datos almacenados
Object.keys(localStorage).forEach(clave => {
    const valor = localStorage.getItem(clave);
    console.log(clave + ": " + valor);
});

// Eliminar un dato concreto
localStorage.removeItem("edad");

// Eliminar todo el almacenamiento
localStorage.clear();
```

Almacenamiento de objetos

El almacenamiento web **no permite guardar objetos directamente**.

Para almacenar objetos es necesario convertirlos a texto utilizando **JSON.stringify()**

```
const usuario = { nombre: "Ana", edad: 25 };
localStorage.setItem("usuario", JSON.stringify(usuario));
```

Para recuperar el objeto es necesario convertirlo de nuevo con **JSON.parse()**

```
const usuarioGuardado = JSON.parse(localStorage.getItem("usuario"));
```

Uso de almacenamiento web junto al DOM

El almacenamiento web suele utilizarse junto con el DOM y los eventos para crear aplicaciones dinámicas.

Usos habituales:

- Guardar datos introducidos por el usuario en un formulario.
- Recuperar los datos al recargar la página.
- Mostrar información almacenada de forma dinámica.
- Eliminar datos mediante la interacción del usuario.

De esta forma, el almacenamiento web permite mantener el estado de una aplicación en el navegador.

Manos a la obra: lista persistente

Desarrolla una página web que permita gestionar una lista de elementos (lista de la compra, tareas, videojuegos, etc.) utilizando **localStorage** para que los datos sean persistentes.

Requisitos de la página:

- Campo de texto para introducir nuevos elementos.
- Botón **“Añadir”** para guardar el elemento en localStorage.
- Botón **“Mostrar lista”** para recuperar y mostrar los datos usando el DOM.
- Al hacer **doble clic** sobre un elemento de la lista (cuando esta se muestra por pantalla), debe eliminarse del DOM y de localStorage.
- Botón **“Limpiar lista”** para borrar todos los datos almacenados.

Eventos de almacenamiento

Eventos de almacenamiento

Además de los métodos de almacenamiento, el navegador proporciona un evento llamado **storage** que permite detectar cambios en el almacenamiento web.

Este evento se utiliza principalmente para **sincronizar información entre varias pestañas o ventanas abiertas** del mismo dominio.

Este evento se dispara cuando:

- Se modifica el contenido de **localStorage**.
- El cambio se produce en **otra pestaña o ventana distinta** del mismo dominio. **NO se dispara en la misma pestaña** en la que se realiza el cambio.

Para escuchar este evento se debe poner un listener para el evento **storage** en el objeto **window.**

El evento **storage** se utiliza principalmente para:

- Sincronizar datos entre pestañas abiertas
- Actualizar interfaces automáticamente
- Detectar cambios externos en los datos almacenados

Eventos de almacenamiento

Cuando se produce el evento **storage**, el navegador proporciona información útil sobre el cambio realizado dentro de las propiedades del objeto **event**:

- **key**: hace referencia al identificador o a la clave del elemento que se ha insertado o borrado del almacenamiento. En el caso de borrar todo el almacén de datos invocando a la función clear(), la clave será **null**.
- **newValue**: contendrá el nuevo valor del elemento a almacenar. En el caso de un evento de borrado, este campo estará vacío.
- **oldValue**: contendrá el valor previo en el almacén de datos. Si se trata de una nueva clave, esta propiedad estará vacía.
- **storageArea**: hace referencia al objeto utilizado para el almacenamiento.
- **url**: se refiere al documento cuyo *script* ha realizado el cambio de almacenamiento.

Eventos de almacenamiento

```
window.addEventListener("storage", function (event) {
  console.log("Clave modificada:", event.key);
  console.log("Valor anterior:", event.oldValue);
  console.log("Nuevo valor:", event.newValue);
});
```

Seguridad y limitaciones

Seguridad del almacenamiento web

El almacenamiento web permite guardar información en el navegador, pero **NO está pensado para almacenar datos sensibles ni críticos.**

Es importante conocer sus riesgos y limitaciones para utilizarlo de forma correcta:

- Los datos **no están cifrados por defecto**.
- Son accesibles desde las herramientas de desarrollo del navegador
- Cualquier script que se ejecute en le página puede acceder a ellos.

Por lo tanto, **NUNCA se deben guardar en localStorage o sessionStorage:**

- Contraseñas
- Datos bancarios
- Tokens de autenticación sensibles
- Información personal crítica

Limitaciones del almacenamiento web

Principales limitaciones del almacenamiento web:

- Posee un espacio limitado: entre 5 y 10 MB, según el navegador
- Sólo permite que se almacenen cadenas de texto
- Dependencia del navegador y del dispositivo
- El usuario puede borrar los datos manualmente

Cookies

¿Qué son las cookies?

Las **cookies** son pequeños fragmentos de información que el navegador almacena y envía automáticamente al servidor en cada petición HTTP.

Algunas consideraciones:

- En el lado del cliente, el código JavaScript en la página web puede acceder y modificar cookies almacenadas en el navegador.
- Las operaciones comunes incluyen:
  - **Creación** de nuevas cookies
  - **Lectura** de valores existentes
  - **Eliminación** de cookies.
- Las cookies pueden ser manipuladas directamente utilizando el objeto **document.cookie** en JavaScript.

Se utilizan principalmente para:

- Gestión de sesiones
- Autenticacion de usuarios
- Seguimiento del usuario

Cookies vs Almacenamiento web

Cookies:

- Se envían al servidor en cada petición.
- Poco espacio disponible (4kB por cookie aproximadamente)
- Uso orientado al backend

localStorage / sessionStorage:

- Sólo accesibles desde JavaScript
- Mayor capacidad de almacenamiento (de 5 a 10 MB)
- Uso orientado al frontend

Creación de cookies

```
// Función para establecer una cookie
function setCookie(name, value, timelife = null){
    // Construir la cadena de la cookie con el nombre y valor
    let cookie = `${name}=${encodeURIComponent(value)}`;

    // Si se proporciona un tiempo de vida, agregar el atributo 'max-age'
    if(timelife !== null){
        // Convertir el tiempo de vida a segundos y agregar al atributo 'max-age'
        cookie += `; max-age=${timelife * 60 * 60 * 24}`;
    }

    // Establecer la cookie en el navegador
    document.cookie = cookie;
}

// Ejemplo de uso:
// Establecer una cookie llamada 'miCookie' con valor 'miValor' que expira en 7 días
setCookie('miCookie', 'miValor', 7);
```

Bases de datos en cliente

Bases de datos en entorno cliente

JavaScript permite usar una base de datos en el cliente para almacenar los datos.

Se usa el objeto **IndexedDB.**

Este objeto permite almacenar una base de datos:

- **No relacional:** sin esquemas ni organización ni tablas, usando pares de clave-valor
- **Persistente**
- **Orientada a objetos**

La ventaja con respecto a localStorage y sessionStorage es que es más potente, eficaz y robusto.

Es más complejo de programar y su uso está menos extendido ya que para el almacenamiento en BBDD suele ser más en la parte de servidor.

Por ello en la mayoría del almacenamiento web se usa localStorage y sessionStorage por ser simples y no requerir operaciones complejas.

Aplicaciones en caché

Aplicaciones en caché

Las aplicaciones web normalmente necesitan conexión a internet para poder funcionar. Sin embargo, es posible usar un mecanismo de caché para ejecutar aplicaciones web sin conexión permanente.

La **caché** permite:

- Almacenar archivos estáticos
- Reducir el número de peticiones al servidor
- Mejorar el rendimiento de la aplicación
- Mejorar la experiencia del usuario.

Para realizar un funcionamiento sin conexión debemos descargar todo el contenido y los recursos de la página previamente. Para hacer esto, se utiliza el archivo **manifest** de HTML5.

Cuando un navegador conecta con una página, lo primero que hace es buscar el archivo y descargar el contenido que hay indicado en él.

El archivo manifest debe tener extensión **“.appcache”**

Aplicaciones en caché

```html
<!DOCTYPE html>
<html manifest="cache.appcache">
<head>
    <link rel="stylesheet" href="styles.css">
    <title>Almacenamiento en cache</title>
</head>
<body>
    Prueba del archivo manifest
    <script src="test.js"></script>
</body>
</html>
```

cache.appcache

```
CACHE MANIFEST
/styles.css
/test.js
/test.png
```