Contenido

[1. Gestionar datos en el almacén 4](#_Toc187854254)

[1.1. Los métodos sessionStorage.setItem y sessionStorage.getItem 4](#_Toc187854255)

[1.2. Ejemplo Completo 5](#_Toc187854256)

[1.3. Trabajar con objetos 6](#_Toc187854257)

[1.4. Diferencias importantes 7](#_Toc187854258)

[1.5. Errores comunes 7](#_Toc187854259)

[2. Recorrer claves 7](#_Toc187854260)

[2.1. Usando un bucle for con la propiedad length 7](#_Toc187854261)

[2.2. Usando un bucle for...in 7](#_Toc187854262)

[2.3. Usando Object.keys() 8](#_Toc187854263)

[2.4. Usando Object.entries() 8](#_Toc187854264)

[2.5. Ejemplo Completo 8](#_Toc187854265)

[2.6. Salida esperada 9](#_Toc187854266)

[3. Almacén local. localStorage 9](#_Toc187854267)

[3.1. Características principales de localStorage 9](#_Toc187854268)

[3.2. Métodos principales de localStorage 9](#_Toc187854269)

[3.3. Ejemplos de uso 9](#_Toc187854270)

[3.4. Ejemplo práctico: Contador de visitas 11](#_Toc187854271)

[3.5. Diferencias con sessionStorage 11](#_Toc187854272)

[3.6. Casos de uso comunes 11](#_Toc187854273)

[4. Borrar datos del almacén 11](#_Toc187854274)

[4.1. Eliminar datos en sessionStorage 11](#_Toc187854275)

[4.2. Eliminar datos en localStorage 12](#_Toc187854276)

[4.3. Ejemplo combinado 13](#_Toc187854277)

[5. El evento storage 13](#_Toc187854278)

[5.1. Características principales del evento storage 14](#_Toc187854279)

[5.2. Cómo usar el evento storage 14](#_Toc187854280)

[5.3. Ejemplo práctico 14](#_Toc187854281)

[5.4. Usos comunes del evento storage 15](#_Toc187854282)

[5.5. Limitaciones del evento storage 16](#_Toc187854283)

**Web Storage**

**Web Storage** es una API de almacenamiento en el navegador web que permite a las aplicaciones guardar datos de manera sencilla y persistente. Es una alternativa más moderna y sencilla en comparación con las cookies, diseñada específicamente para manejar cantidades de datos más grandes y evitar algunas limitaciones inherentes a las cookies.

**Tipos de Web Storage**

Web Storage se divide en dos tipos principales:

1. **Local Storage (localStorage)**:
   * Los datos almacenados son persistentes, lo que significa que permanecen disponibles incluso después de cerrar el navegador o apagar el dispositivo.
   * Los datos solo se eliminan manualmente por el usuario o mediante un script.
   * Se utiliza para almacenar datos que deben estar disponibles en diferentes sesiones de navegación.
2. **Session Storage (sessionStorage)**:
   * Los datos almacenados solo están disponibles durante la sesión del navegador.
   * Una vez que se cierra la pestaña o ventana del navegador, los datos se eliminan automáticamente.
   * Es útil para almacenar datos temporales o específicos de la sesión actual.

**Características de Web Storage**

* **Capacidad**: Puede almacenar varios megabytes de datos (generalmente alrededor de 5 MB por dominio), mucho más que las cookies.
* **Scope**: Los datos están limitados al dominio (y protocolo) que los creó.
* **Facilidad de uso**: La API es sencilla y consistente, con métodos para establecer, obtener y eliminar datos.
* **Seguridad**: Los datos no se envían automáticamente al servidor en cada solicitud HTTP como sucede con las cookies.

**Métodos Comunes**

Ambos tipos de almacenamiento (localStorage y sessionStorage) comparten una API similar.

1. **Guardar un dato**:

localStorage.setItem('clave', 'valor');

sessionStorage.setItem('usuario', 'Juan');

1. **Obtener un dato**:

const valor = localStorage.getItem('clave'); // "valor"

const usuario = sessionStorage.getItem('usuario'); // "Juan"

1. **Eliminar un dato**:

localStorage.removeItem('clave');

sessionStorage.removeItem('usuario');

1. **Eliminar todos los datos**:

localStorage.clear();

sessionStorage.clear();

1. **Iterar sobre los datos almacenados**:

for (let i = 0; i < localStorage.length; i++) {

const key = localStorage.key(i);

console.log(`${key}: ${localStorage.getItem(key)}`);

}

**Ejemplo Práctico**

**Guardar y recuperar el estado de un tema oscuro o claro**:

// Guardar la preferencia del usuario en localStorage

function toggleTheme(isDarkMode) {

localStorage.setItem('theme', isDarkMode ? 'dark' : 'light');

}

// Obtener la preferencia del usuario

function loadTheme() {

const theme = localStorage.getItem('theme');

if (theme === 'dark') {

document.body.classList.add('dark-mode');

} else {

document.body.classList.remove('dark-mode');

}

}

// Cambiar tema

document.getElementById('theme-toggle').addEventListener('click', () => {

const isDarkMode = document.body.classList.toggle('dark-mode');

toggleTheme(isDarkMode);

});

// Aplicar tema al cargar la página

loadTheme();

**Diferencias con Cookies**

|  |  |  |
| --- | --- | --- |
| Característica | Web Storage | Cookies |
| Tamaño máximo | ~5 MB | ~4 KB |
| Envío automático al servidor | No | Sí (con cada solicitud) |
| Persistencia | Local o de sesión | Depende de la configuración |
| API | Sencilla y moderna | Más compleja |

**Usos Comunes**

* Guardar configuraciones del usuario (tema, idioma, etc.).
* Almacenar datos temporales como un carrito de compras.
* Manejar sesiones de usuario sin usar cookies (con sessionStorage).

Web Storage es una herramienta poderosa para manejar datos del lado del cliente, simplificando el desarrollo y mejorando la experiencia del usuario. 🎉

### Gestionar datos en el almacén

### Los métodos sessionStorage.setItem y sessionStorage.getItem

Los métodos **sessionStorage.setItem** y **sessionStorage.getItem** son parte de la API de **Web Storage** en JavaScript y están diseñados para trabajar con el almacenamiento de sesión del navegador, conocido como **sessionStorage**. Este almacenamiento es temporal y está limitado a la duración de la pestaña o ventana del navegador.

**sessionStorage.setItem**

Este método se utiliza para guardar un par clave-valor en el almacenamiento de sesión.

#### **Sintaxis**

sessionStorage.setItem(key, value);

* **key**: Una cadena de texto que representa el nombre de la clave.
* **value**: El valor asociado con la clave. Este valor debe ser una cadena de texto; si es de otro tipo, como un objeto o número, se debe convertir a texto, por ejemplo, usando JSON.stringify.

#### **Ejemplo**

Guardar información en sessionStorage:

sessionStorage.setItem('usuario', 'Juan');

sessionStorage.setItem('rol', 'administrador')

**sessionStorage.getItem**

Este método se utiliza para obtener el valor asociado a una clave específica en el almacenamiento de sesión.

#### **Sintaxis**

const value = sessionStorage.getItem(key);

* **key**: Una cadena de texto que representa el nombre de la clave cuyo valor se desea obtener.
* **value**: Devuelve el valor asociado con la clave como una cadena de texto. Si la clave no existe, devuelve null.

#### **Ejemplo**

Recuperar información desde sessionStorage:

const usuario = sessionStorage.getItem('usuario'); // "Juan"

const rol = sessionStorage.getItem('rol'); // "administrador"

console.log(`Usuario: ${usuario}, Rol: ${rol}`);

### Ejemplo Completo

Guardar y recuperar datos del almacenamiento de sesión:

// Guardar datos

sessionStorage.setItem('nombre', 'María');

sessionStorage.setItem('edad', '30');

// Recuperar datos

const nombre = sessionStorage.getItem('nombre'); // "María"

const edad = sessionStorage.getItem('edad'); // "30"

console.log(`Nombre: ${nombre}, Edad: ${edad}`);

// Comprobar datos inexistentes

const ciudad = sessionStorage.getItem('ciudad'); // null

console.log(ciudad); // null

### Trabajar con objetos

Dado que sessionStorage almacena solo cadenas, debes convertir los objetos a formato JSON al guardarlos y volver a convertirlos al recuperarlos.

#### **Guardar un objeto**

const usuario = {

nombre: 'Carlos',

edad: 25,

rol: 'editor'

};

// Convertir objeto a JSON y guardarlo

sessionStorage.setItem('usuario', JSON.stringify(usuario));

#### **Recuperar un objeto**

// Recuperar el objeto como JSON

const usuarioJSON = sessionStorage.getItem('usuario');

// Convertir JSON de nuevo a un objeto

const usuarioObj = JSON.parse(usuarioJSON);

console.log(usuarioObj.nombre); // "Carlos"

console.log(usuarioObj.edad); // 25

### Diferencias importantes

* Los datos guardados con sessionStorage son específicos de la pestaña/ventana del navegador y no están disponibles en otras pestañas, incluso si pertenecen al mismo dominio.
* Si cierras la pestaña o ventana, todos los datos en sessionStorage se eliminan automáticamente.

### Errores comunes

1. **No convertir datos complejos**: Si intentas guardar un objeto sin convertirlo a JSON, obtendrás un error o comportamiento inesperado.

sessionStorage.setItem('usuario', { nombre: 'Ana' }); // Incorrecto

1. **No verificar claves inexistentes**: Si intentas obtener un valor de una clave que no existe, sessionStorage.getItem devuelve null.

const valor = sessionStorage.getItem('claveInexistente'); // null

### Recorrer claves

Para recorrer las claves almacenadas en el objeto sessionStorage, puedes utilizar diferentes métodos en JavaScript.

### Usando un bucle for con la propiedad length

El objeto sessionStorage tiene una propiedad length que indica cuántas claves hay almacenadas. Puedes usar un bucle for junto con el método key() para acceder a cada clave.

for (let i = 0; i < sessionStorage.length; i++) {

const clave = sessionStorage.key(i); // Obtiene la clave en la posición i

const valor = sessionStorage.getItem(clave); // Obtiene el valor asociado

console.log(`Clave: ${clave}, Valor: ${valor}`);

}

### Usando un bucle for...in

Aunque sessionStorage no es técnicamente un objeto iterable estándar, puedes usar un bucle for...in para iterar sobre las claves almacenadas.

for (const clave in sessionStorage) {

if (sessionStorage.hasOwnProperty(clave)) { // Verifica que sea una clave propia

const valor = sessionStorage.getItem(clave); // Obtiene el valor asociado

console.log(`Clave: ${clave}, Valor: ${valor}`);

}

}

**Nota:** El uso de hasOwnProperty es importante para evitar propiedades heredadas de sessionStorage.

### Usando Object.keys()

Puedes usar Object.keys() para obtener un array con todas las claves de sessionStorage y luego recorrerlas.

const claves = Object.keys(sessionStorage);

claves.forEach(clave => {

const valor = sessionStorage.getItem(clave);

console.log(`Clave: ${clave}, Valor: ${valor}`);

});

### Usando Object.entries()

Si deseas recorrer las claves junto con sus valores directamente, puedes usar Object.entries().

const pares = Object.entries(sessionStorage);

pares.forEach(([clave, valor]) => {

console.log(`Clave: ${clave}, Valor: ${valor}`);

});

### Ejemplo Completo

// Agregar algunas claves al sessionStorage

sessionStorage.setItem("usuario", "Carlos");

sessionStorage.setItem("rol", "administrador");

sessionStorage.setItem("tema", "oscuro");

// Recorrer las claves

for (let i = 0; i < sessionStorage.length; i++) {

const clave = sessionStorage.key(i);

const valor = sessionStorage.getItem(clave);

console.log(`Clave: ${clave}, Valor: ${valor}`);

}

### Salida esperada

Si tienes las claves y valores como en el ejemplo, la consola mostrará:

Clave: usuario, Valor: Carlos

Clave: rol, Valor: administrador

Clave: tema, Valor: oscuro

### Almacén local. localStorage

localStorage es una API de almacenamiento web que permite guardar datos clave-valor en el navegador de forma persistente. A diferencia de sessionStorage, los datos en localStorage no tienen una fecha de expiración, por lo que permanecen incluso después de cerrar el navegador o reiniciar el dispositivo.

### Características principales de localStorage

1. **Capacidad de almacenamiento**: Hasta 5 MB (varía según el navegador).
2. **Persistencia**: Los datos se conservan hasta que se eliminen manualmente o a través de código.
3. **Acceso síncrono**: Los métodos de localStorage no implican promesas ni operaciones asíncronas.

### Métodos principales de localStorage

1. **setItem(key, value)**: Guarda un valor asociado a una clave.
2. **getItem(key)**: Recupera el valor asociado a una clave.
3. **removeItem(key)**: Elimina una clave y su valor.
4. **clear()**: Elimina todos los datos almacenados.
5. **key(index)**: Devuelve el nombre de la clave en una posición específica.
6. **length**: Devuelve el número total de claves almacenadas.

### Ejemplos de uso

#### **1. Guardar datos en** localStorage

localStorage.setItem("usuario", "Carlos");

localStorage.setItem("rol", "administrador");

console.log("Datos guardados en localStorage.");

Resultado: Los datos se guardan en el navegador y se pueden ver en la consola de desarrollo (Aplicación > Almacenamiento > localStorage).

#### **2. Leer datos de** localStorage

const usuario = localStorage.getItem("usuario");

const rol = localStorage.getItem("rol");

if (usuario && rol) {

console.log(`Usuario: ${usuario}, Rol: ${rol}`);

} else {

console.log("No hay datos disponibles.");

}

Salida esperada:

Usuario: Carlos, Rol: administrador

#### **3. Eliminar un elemento específico**

localStorage.removeItem("rol");

console.log("El rol fue eliminado de localStorage.");

#### **4. Limpiar todos los datos**

localStorage.clear();

console.log("Todos los datos de localStorage han sido eliminados.");

#### **5. Recorrer todas las claves**

localStorage.setItem("usuario", "Carlos");

localStorage.setItem("tema", "oscuro");

localStorage.setItem("idioma", "español");

for (let i = 0; i < localStorage.length; i++) {

const clave = localStorage.key(i);

const valor = localStorage.getItem(clave);

console.log(`Clave: ${clave}, Valor: ${valor}`);

}

Salida esperada:

Clave: usuario, Valor: Carlos

Clave: tema, Valor: oscuro

Clave: idioma, Valor: español

### Ejemplo práctico: Contador de visitas

// Incrementar el contador de visitas cada vez que se abre la página

const visitas = localStorage.getItem("visitas") || 0;

localStorage.setItem("visitas", parseInt(visitas) + 1);

console.log(`Has visitado esta página ${localStorage.getItem("visitas")} veces.`);

Salida esperada (en la consola del navegador):

Has visitado esta página 1 veces.

Has visitado esta página 2 veces.

...

### Diferencias con sessionStorage

|  |  |  |
| --- | --- | --- |
| Característica | localStorage | sessionStorage |
| Persistencia | Los datos se conservan indefinidamente. | Los datos se eliminan al cerrar la pestaña o navegador. |
| Propósito | Almacenar datos de larga duración. | Almacenar datos temporales de una sesión. |
| Capacidad | ~5 MB (varía según navegador). | Igual a localStorage. |

### Casos de uso comunes

1. **Preferencias del usuario**: Tema oscuro, idioma, configuraciones.
2. **Carrito de compras**: Guardar productos seleccionados.
3. **Estado de la sesión**: Mantener información del usuario entre sesiones.

### Borrar datos del almacén

Tanto sessionStorage como localStorage ofrecen métodos para eliminar datos.

### Eliminar datos en sessionStorage

1. **Eliminar una clave específica usando removeItem()**

sessionStorage.setItem("usuario", "Carlos");

sessionStorage.setItem("tema", "oscuro");

// Eliminar solo la clave 'tema'

sessionStorage.removeItem("tema");

console.log("La clave 'tema' ha sido eliminada.");

Resultado: Solo la clave tema es eliminada, pero usuario permanece en sessionStorage.

1. **Eliminar todos los datos usando clear()**

sessionStorage.setItem("usuario", "Carlos");

sessionStorage.setItem("tema", "oscuro");

// Limpiar todos los datos de sessionStorage

sessionStorage.clear();

console.log("Todos los datos de sessionStorage han sido eliminados.");

Resultado: Todos los datos almacenados en sessionStorage se eliminan.

### Eliminar datos en localStorage

1. **Eliminar una clave específica usando removeItem()**

localStorage.setItem("usuario", "Carlos");

localStorage.setItem("tema", "oscuro");

// Eliminar solo la clave 'usuario'

localStorage.removeItem("usuario");

console.log("La clave 'usuario' ha sido eliminada.");

Resultado: Solo la clave usuario es eliminada, pero tema permanece en localStorage.

1. **Eliminar todos los datos usando clear()**

localStorage.setItem("usuario", "Carlos");

localStorage.setItem("tema", "oscuro");

// Limpiar todos los datos de localStorage

localStorage.clear();

console.log("Todos los datos de localStorage han sido eliminados.");

Resultado: Todos los datos almacenados en localStorage se eliminan.

### Ejemplo combinado

Aquí tienes un ejemplo que combina ambos tipos de almacenamiento:

// Agregar datos a ambos tipos de almacenamiento

localStorage.setItem("carrito", "productos");

sessionStorage.setItem("sesionActiva", "true");

// Eliminar un dato de localStorage

localStorage.removeItem("carrito");

console.log("Se eliminó 'carrito' de localStorage.");

// Limpiar todo en sessionStorage

sessionStorage.clear();

console.log("Todos los datos de sessionStorage han sido eliminados.");

**Diferencias entre removeItem() y clear()**

* **removeItem()**: Permite eliminar una clave y su valor específicos.
* **clear()**: Borra todos los datos del almacenamiento correspondiente.

**Comprobando si los datos fueron eliminados**

Puedes verificar si las claves existen después de eliminarlas:

localStorage.setItem("usuario", "Carlos");

sessionStorage.setItem("sesion", "activa");

// Eliminar las claves

localStorage.removeItem("usuario");

sessionStorage.clear();

// Verificar los resultados

console.log(localStorage.getItem("usuario")); // null

console.log(sessionStorage.getItem("sesion")); // null

Si las claves han sido eliminadas correctamente, el resultado será null.

### El evento storage

El **evento storage** en localStorage se activa cuando ocurre un cambio en el almacenamiento de un dominio, siempre y cuando el cambio sea realizado en una pestaña o ventana diferente a la que escucha el evento.

### Características principales del evento storage

1. **Dónde se activa:**
   * Solo se activa en otras pestañas, ventanas o iframes que compartan el mismo origen (protocolo, dominio y puerto).
   * No se activa en la misma pestaña que realiza el cambio en localStorage.
2. **Qué lo dispara:**
   * Añadir una clave: localStorage.setItem("clave", "valor").
   * Modificar el valor de una clave existente.
   * Eliminar una clave: localStorage.removeItem("clave").
   * Borrar todo el almacenamiento: localStorage.clear().
3. **Propiedades del evento storage:**
   * **key**: Nombre de la clave que cambió.
   * **oldValue**: Valor previo de la clave.
   * **newValue**: Nuevo valor de la clave.
   * **url**: URL de la página que realizó el cambio.
   * **storageArea**: Referencia al almacenamiento que se modificó (generalmente localStorage).

### Cómo usar el evento storage

Para escuchar el evento storage, debes usar el método addEventListener en el objeto window.

window.addEventListener("storage", (event) => {

console.log("Cambio detectado en localStorage:");

console.log(`Clave: ${event.key}`);

console.log(`Valor anterior: ${event.oldValue}`);

console.log(`Nuevo valor: ${event.newValue}`);

console.log(`Origen del cambio: ${event.url}`);

console.log(`Área de almacenamiento: ${event.storageArea}`);

});

### Ejemplo práctico

#### **1. Configurar el listener del evento**

En una pestaña o ventana del navegador:

window.addEventListener("storage", (event) => {

console.log(`La clave '${event.key}' cambió.`);

console.log(`Valor anterior: ${event.oldValue}`);

console.log(`Nuevo valor: ${event.newValue}`);

console.log(`Cambio originado desde: ${event.url}`);

});

#### **2. Modificar el** localStorage **desde otra pestaña**

En otra pestaña, ejecuta lo siguiente:

localStorage.setItem("usuario", "Carlos"); // Agrega una clave

localStorage.setItem("usuario", "María"); // Modifica la clave

localStorage.removeItem("usuario"); // Elimina la clave

#### **Salida esperada en la primera pestaña:**

La clave 'usuario' cambió.

Valor anterior: null

Nuevo valor: Carlos

Cambio originado desde: http://ejemplo.com/

La clave 'usuario' cambió.

Valor anterior: Carlos

Nuevo valor: María

Cambio originado desde: http://ejemplo.com/

La clave 'usuario' cambió.

Valor anterior: María

Nuevo valor: null

Cambio originado desde: http://ejemplo.com/

### Usos comunes del evento storage

1. **Sincronización entre pestañas o ventanas:**
   * Mantener un estado compartido, como un tema (oscuro o claro) o el idioma del usuario.
2. **Cerrar sesión en múltiples pestañas:**
   * Si un usuario cierra sesión en una pestaña, puedes detectar el cambio en otras pestañas y redirigirlas a la página de inicio.

window.addEventListener("storage", (event) => {

if (event.key === "isLoggedIn" && event.newValue === "false") {

console.log("Cierre de sesión detectado. Redirigiendo...");

window.location.href = "/login";

}

});

1. **Notificaciones de cambios:**
   * Informar al usuario sobre cambios realizados en otra pestaña.

### Limitaciones del evento storage

1. **No funciona con sessionStorage:**
   * El evento storage solo aplica a localStorage.
2. **No se activa en la pestaña que realiza el cambio:**
   * Si cambias un valor en la misma pestaña, el evento no se disparará.