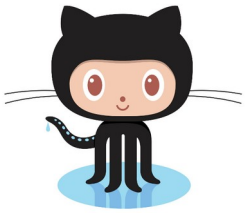
Desarrollo web en entorno cliente

UD – 5

WebStorage

Contenidos



ÍNDICE

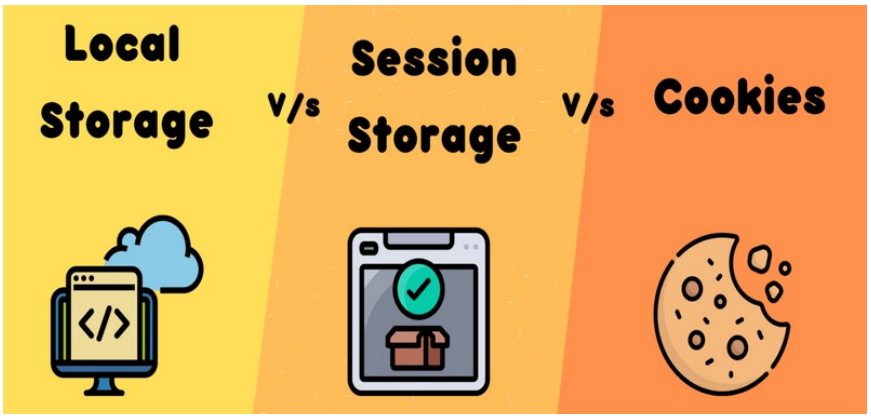
1. INTRODUCCIÓN

localStorage y **sessionStorage** son herramientas del estándar **Web Storage** que permiten almacenar datos en el navegador en forma de pares clave-valor. Estas APIs son rápidas y fáciles de usar, ideales para guardar información del usuario sin necesidad de recurrir al servidor.

La principal diferencia entre ambas es su persistencia:

- los datos en localStorage permanecen incluso al cerrar el navegador.
- sessionStorage se eliminan al cerrar la pestaña o ventana.

Ambas son útiles para mejorar la experiencia del usuario, permitiendo conservar configuraciones, estado de la aplicación o datos temporales. Su simplicidad las hace esenciales en el desarrollo web moderno.



2. WEBSTORAGE

El almacenamiento web proporciona una manera eficiente y sencilla de almacenar datos directamente en el navegador del usuario. Dos de las principales APIs para este propósito son **localStorage** y **sessionStorage**, que forman parte del estándar **Web Storage**. Estas herramientas permiten guardar pares clave-valor que persisten entre las solicitudes de red, pero tienen diferentes características en cuanto a su alcance y duración.

Clave-Valor

- Ambos almacenamientos funcionan guardando datos en forma de pares clave-valor, donde tanto la clave como el valor son cadenas de texto.
- Si necesitas almacenar datos complejos (como objetos), puedes convertirlos a JSON usando JSON.stringify y recuperarlos con JSON.parse.

Tamaño

- La capacidad de almacenamiento varía según el navegador, pero generalmente tienen un límite de 5-10 MB.

Persistencia

- **localStorage**:
  - Los datos permanecen almacenados indefinidamente, incluso si el usuario cierra el navegador o apaga el dispositivo.
  - Es ideal para guardar configuraciones de usuario o datos que deben persistir entre sesiones.
- **sessionStorage**:
  - Los datos se eliminan automáticamente cuando se cierra la pestaña o ventana del navegador.
  - Es útil para almacenar información que solo es relevante durante la sesión actual, como datos

temporales o de formularios.

Por lo tanto, resumiendo los aspectos anteriores:

Cacterísticas	LocalStorage	SesionStorage
Duración	*Persistente hasta que se borre manualmente.*	*Solo dura mientras la pestaña esté abierta.*
Alcance de la sesión	*Compartido entre todas las pestañas del mismo origen.*	*Limitado a la pestaña o ventana actual.*
Uso típico	*Configuraciones, temas, preferencias, tokens de autenticación.*	*Datos temporales, estado de un formulario.*

Y la forma de usuarlo en Vanilla Javascript:

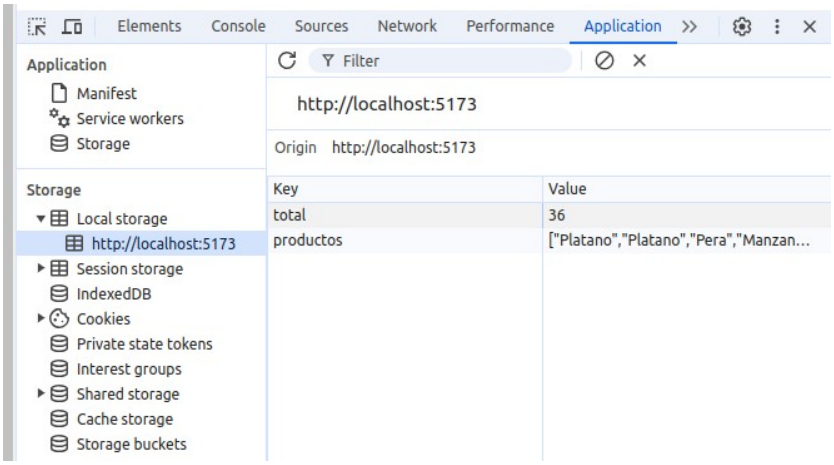
- **Guardar un valor**
```
localStorage.setItem("nombre", "Juan");
sessionStorage.setItem("edad", "30");
localStorage.setItem("usuario", JSON.stringify({ nombre: "Juan", edad: 30 }));
```

- **Leer un valor**
```
const nombre = localStorage.getItem("nombre");
const edad = sessionStorage.getItem("edad");
const usuarioGuardado = JSON.parse(localStorage.getItem("usuario"));
```

- **Eliminar un valor**
```
localStorage.removeItem("nombre");
sessionStorage.removeItem("edad");
```

- **Limpiar todo el almacenamiento**
```
localStorage.clear();
sessionStorage.clear();
```

Si accedes a la consola podrás ver toda la información almacenada en el navegador:



3. SERVICIO DE LOCALSTORAGE

El servicio **LocalStorageServicio** es una clase utilitaria diseñada para simplificar y centralizar las operaciones comunes con localStorage en una aplicación. Proporciona métodos estáticos que encapsulan la lógica de lectura, escritura, eliminación y limpieza del almacenamiento local, manejando automáticamente la serialización y deserialización de datos JSON y los posibles errores.

Ventajas

- ✔ **Abstracción de lógica repetitiva:**
  - Centraliza las operaciones con localStorage, evitando escribir el mismo código en diferentes partes de la aplicación.
- ✔ **Manejo de errores**
  - Incluye bloques try-catch para capturar y registrar errores que puedan ocurrir, mejorando la confiabilidad.
- ✔ **Facilidad de uso**
  - Los métodos son simples y claros, con nombres que describen su propósito
- ✔ **Soporte para datos complejos**
  - Serializa y deserializa automáticamente los valores, permitiendo trabajar con objetos y arrays sin esfuerzo adicional.

```
class LocalStorageServicio {
    static get(valor) {
      try {
        const item = window.localStorage.getItem(valor);
        return item ? JSON.parse(item) : null;
      } catch (error) {
        console.error("Error LEYENDO el valor", valor, error);
        return null;
      }
    }

    static set(clave, valor) {
      try {
        window.localStorage.setItem(clave, JSON.stringify(valor));
      } catch (error) {
        console.error("Error GUARDANDO el valor", clave, error);
      }
    }

    static remove(valor) {
      try {
        window.localStorage.removeItem(valor);
      } catch (error) {
        console.error("Error BORRANDO el valor", valor, error);
      }
    }

    static clear() {
      try {
        window.localStorage.clear();
      } catch (error) {
        console.error("Error LIMPIANDO localStorage", error);
      }
    }
}

export default LocalStorageServicio;
```

4. ALMACENAR VARIABLES ESTADO

Este es un **custom hook** en React llamado **UseStorageState**, que combina la funcionalidad de React con localStorage para gestionar el estado de manera persistente. Es decir, permite guardar y recuperar datos automáticamente desde localStorage, lo que hace que el estado persista incluso después de recargar la página. Este nuevo hook va a centralizar el uso de 3 importantes funcionalidades:

- **useState** → para gestionar el estado
- **useEffect** → los efectos secundarios(almacenar el estado al cambiar una variable).
- **LocalStorageServicio** → para almacenar la información en el "navegador"

Ventajas

✔ **Persistencia automática**

- Los datos del estado persisten incluso después de recargar la página o cerrar el navegador.

✔ **Reutilizable**

- Este hook puede ser usado en cualquier componente simplemente importándolo y pasando la clave y el valor inicial.

✔ **Abstracción limpia**

- La lógica de interacción con localStorage está encapsulada dentro del hook y el servicio, manteniendo el código de los componentes más limpio.

```
import { useState, useEffect } from "react";
import LocalStorageServicio from "./storage.js"

function UseStorageState(clave, valorInicial) {
  const [state, setState] = useState(() => {

    // Recupera el valor inicial desde localStorage o usa el valor predeterminado
    const valorGuardado = LocalStorageServicio.get(clave);
    return valorGuardado !== null ? valorGuardado : valorInicial;
  });

  useEffect(() => {

    // Guarda el estado en localStorage cada vez que cambie
    LocalStorageServicio.set(clave, state);

  }, [clave, state]);

  return [state, setState];
}

export default UseStorageState;
```