[1. Las cookies 2](#_Toc187836146)

[1.1. Características clave de las cookies 3](#_Toc187836147)

[1.2. Usos comunes de las cookies 3](#_Toc187836148)

[1.3. Cómo funcionan las cookies 3](#_Toc187836149)

[1.4. Ejemplo de encabezado HTTP 4](#_Toc187836150)

[1.5. Seguridad en el uso de cookies 4](#_Toc187836151)

[1.6. Ventajas y desventajas de las cookies 4](#_Toc187836152)

[2. El objeto document.cookie 4](#_Toc187836153)

[2.1. Características principales de document.cookie: 4](#_Toc187836154)

[2.2. Sintaxis básica para escribir cookies 5](#_Toc187836155)

[2.3. Ejemplos 5](#_Toc187836156)

[2.4. Consideraciones importantes 6](#_Toc187836157)

[2.5. Ejercicio práctico 6](#_Toc187836158)

[3. La función setCookie() 6](#_Toc187836159)

[3.1. Ejemplo de función setCookie() 6](#_Toc187836160)

[3.2. Parámetros explicados 7](#_Toc187836161)

[3.3. Uso básico de setCookie() 7](#_Toc187836162)

[3.4. Extender la función setCookie() con más opciones 7](#_Toc187836163)

[3.5. Uso de la función extendida 8](#_Toc187836164)

[3.6. Notas importantes 8](#_Toc187836165)

[4. La función getCookie() 9](#_Toc187836166)

[4.1. Cómo funciona document.cookie 9](#_Toc187836167)

[4.2. Implementación básica de getCookie() 9](#_Toc187836168)

[4.3. Uso básico de getCookie() 9](#_Toc187836169)

[4.4. Mejorando getCookie() 10](#_Toc187836170)

[4.5. Ejemplo de uso avanzado 10](#_Toc187836171)

[4.6. Notas importantes 10](#_Toc187836172)

[5. Eliminar cookies 11](#_Toc187836173)

[5.1. Cómo funcionan las cookies y su expiración 11](#_Toc187836174)

[5.2. Métodos para borrar cookies 11](#_Toc187836175)

[5.3. Función para borrar cookies 11](#_Toc187836176)

[5.4. Notas importantes al eliminar cookies 12](#_Toc187836177)

[5.5. Eliminar todas las cookies 12](#_Toc187836178)

[5.6. Extensión para manejar domain y Secure 12](#_Toc187836179)

[5.7. Conclusión 13](#_Toc187836180)

**Almacenamiento de información en el navegador. Cookies**

En el desarrollo web moderno, las aplicaciones necesitan gestionar datos de forma eficiente para mejorar la experiencia del usuario. Una solución fundamental es el almacenamiento de información en el navegador, que permite a las aplicaciones guardar datos localmente en el dispositivo del usuario. Esto no solo reduce la dependencia de servidores remotos, sino que también habilita funcionalidades como la persistencia de información entre sesiones, el trabajo sin conexión y la personalización del contenido.

Existen varias APIs de almacenamiento en el navegador, cada una diseñada para diferentes casos de uso, con características específicas en cuanto a capacidad, seguridad y accesibilidad. Estas incluyen:

1. Cookies: La opción más antigua, utilizada principalmente para rastrear sesiones y transferir información pequeña al servidor.
2. LocalStorage: Almacenamiento clave-valor sincronizado y persistente, ideal para datos simples.
3. SessionStorage: Similar a LocalStorage, pero con duración limitada a la sesión del navegador.
4. IndexedDB: Una base de datos local para almacenar grandes volúmenes de datos estructurados.
5. Cache Storage: Diseñada para manejar recursos de aplicaciones web y trabajar junto con los **Service Workers**.

#### Importancia del almacenamiento en el navegador

* Mejora del rendimiento: Los datos almacenados localmente pueden ser accesibles más rápidamente que los obtenidos de un servidor remoto.
* Soporte offline: Permite a las aplicaciones web funcionar incluso sin conexión a Internet.
* Personalización: Facilita la persistencia de configuraciones y preferencias del usuario.
* Reducción de la carga en el servidor: Minimiza el número de solicitudes al servidor, optimizando el uso de recursos.

En este tema, exploraremos las principales tecnologías de almacenamiento, cómo funcionan, cuándo utilizarlas y las mejores prácticas para garantizar seguridad y eficiencia al manejar datos del usuario.

### **Las cookies**

Las cookies son pequeños fragmentos de datos que un servidor web envía al navegador del usuario y que este almacena para ser utilizados en futuras interacciones con el mismo sitio web. Son una de las formas más antiguas y ampliamente utilizadas de almacenar información en los navegadores, permitiendo que las aplicaciones web mantengan estado y personalización.

### **Características clave de las cookies**

1. Tamaño limitado: Cada cookie puede almacenar un máximo de 4 KB de datos.
2. Asociadas a un dominio: Solo pueden ser accedidas por el dominio que las creó, lo que garantiza cierto nivel de seguridad.
3. Duración definida: Pueden configurarse como:
   * Cookies de sesión: Se eliminan cuando el usuario cierra el navegador.
   * Cookies persistentes: Permanecen almacenadas durante un tiempo definido, incluso después de cerrar el navegador.
4. Envío automático: Se envían automáticamente al servidor con cada solicitud HTTP, como al cargar una página o enviar un formulario.

### **Usos comunes de las cookies**

* Autenticación de usuarios: Almacenar información de sesión para mantener al usuario autenticado en un sitio web.
* Preferencias del usuario: Guardar configuraciones como idioma, tema oscuro o preferencias de visualización.
* Carritos de compra: Rastrear productos seleccionados en sitios de comercio electrónico.
* Analítica web: Recopilar datos sobre el comportamiento de los usuarios en el sitio.
* Publicidad personalizada: Mostrar anuncios basados en el historial de navegación (a menudo mediante cookies de terceros).

### **Cómo funcionan las cookies**

1. Creación: El servidor envía una cookie al navegador mediante el encabezado HTTP Set-Cookie o JavaScript.
2. Almacenamiento: El navegador guarda la cookie en el dispositivo del usuario, asociada al dominio correspondiente.
3. Envío automático: En cada solicitud posterior al mismo dominio, el navegador envía las cookies almacenadas en el encabezado HTTP Cookie.
4. Lectura: El servidor o el cliente (a través de JavaScript) puede leer las cookies para utilizarlas en la aplicación.

### **Ejemplo de encabezado HTTP**

#### Respuesta del servidor:

Set-Cookie: usuario=Juan; Expires=Wed, 15 Jan 2025 12:00:00 GMT; Secure; HttpOnly

#### Solicitud posterior del navegador:

Cookie: usuario=Juan

### **Seguridad en el uso de cookies**

* HttpOnly: Hace que las cookies no sean accesibles desde JavaScript, reduciendo el riesgo de ataques XSS.
* Secure: Indica que las cookies solo se envían a través de conexiones HTTPS.
* SameSite: Limita el envío de cookies en solicitudes de origen cruzado para prevenir ataques CSRF.

### **Ventajas y desventajas de las cookies**

|  |  |
| --- | --- |
| Ventajas | Desventajas |
| Simples de usar y ampliamente soportadas. | Capacidad de almacenamiento limitada (4 KB). |
| Ayudan a mantener el estado entre sesiones. | Se envían automáticamente en cada solicitud, lo que puede aumentar el tráfico de red. |
| **Configurables en duración y alcance.** | **Pueden ser utilizadas para rastrear usuarios, generando preocupaciones de privacidad.** |

Las cookies son una herramienta poderosa, pero en el desarrollo web moderno, a menudo se complementan o reemplazan por otras tecnologías como LocalStorage, SessionStorage e IndexedDB, dependiendo de las necesidades de la aplicación.

### **El objeto document.cookie**

El objeto **document.cookie** en JavaScript se utiliza para trabajar con cookies en el navegador. Las cookies son pequeños fragmentos de texto que los sitios web almacenan en el navegador del usuario para recordar información entre solicitudes o sesiones.

### **Características principales de document.cookie:**

1. Leer cookies: Puedes obtener todas las cookies accesibles al documento como una cadena.
2. Escribir cookies: Puedes agregar o actualizar cookies definiendo su nombre, valor y atributos.
3. Limitaciones:
   * Solo puedes acceder a cookies disponibles en el dominio actual y que no estén marcadas como **HttpOnly**.
   * La capacidad máxima para cookies en el navegador es limitada (alrededor de 4 KB por cookie).

### **Sintaxis básica para escribir cookies**

document.cookie = "nombre=valor; atributos";

Donde los atributos pueden incluir:

* **expires**=fecha: Fecha en la que la cookie expirará.
* **max-age**=segundos: Tiempo en segundos antes de que la cookie expire.
* **path**=ruta: Ruta para la que la cookie será válida.
* **domain**=dominio: Dominio donde la cookie es accesible.
* **secure**: Hace que la cookie solo sea enviada a través de HTTPS.
* **SameSite**: Controla si la cookie será enviada con solicitudes de terceros. Valores: Strict, Lax, None.

### **Ejemplos**

#### 1. Escribir una cookie simple

document.cookie = "usuario=Juan";

console.log(document.cookie); // Salida: "usuario=Juan"

#### 2. Agregar una cookie con fecha de expiración

let fechaExpiracion = new Date();

fechaExpiracion.setDate(fechaExpiracion.getDate() + 7); // 7 días desde hoy

document.cookie = `tema=oscuro; expires=${fechaExpiracion.toUTCString()}`;

console.log(document.cookie); // Salida: "usuario=Juan; tema=oscuro"

#### 3. Especificar una cookie para un path

document.cookie = "idioma=es; path=/subcarpeta";

console.log(document.cookie); // Salida depende de si estás en /subcarpeta o no

#### 4. Leer todas las cookies

console.log(document.cookie);

// Salida: "usuario=Juan; tema=oscuro; idioma=es"

#### 5. Actualizar una cookie

document.cookie = "usuario=Carlos";

console.log(document.cookie); // Salida: "usuario=Carlos; tema=oscuro; idioma=es"

#### 6. Eliminar una cookie

Para eliminar una cookie, establece una fecha de expiración pasada.

document.cookie = "tema=; expires=Thu, 01 Jan 1970 00:00:00 UTC";

console.log(document.cookie); // Salida: "usuario=Carlos; idioma=es"

#### 7. Usar el atributo Secure y SameSite

document.cookie = "token=12345; secure; SameSite=Strict";

console.log(document.cookie); // Salida: "usuario=Carlos; idioma=es; token=12345" (si es HTTPS)

### **Consideraciones importantes**

1. No puedes eliminar directamente una cookie que no tiene los mismos atributos (path, domain) con los que fue creada.
2. Cookies marcadas como HttpOnly no son accesibles desde document.cookie.
3. El navegador gestiona las cookies por dominio, por lo que no podrás acceder a cookies de otros sitios.

### **Ejercicio práctico**

Escribe un script que almacene el nombre del usuario en una cookie y lo recupere cuando el usuario vuelva a visitar la página:

// Establecer la cookie si no existe

if (!document.cookie.includes("nombreUsuario")) {

let nombre = prompt("Por favor, ingresa tu nombre:");

document.cookie = `nombreUsuario=${nombre}; max-age=86400`; // Expira en 1 día

alert("Tu nombre se ha guardado en una cookie.");

} else {

// Leer y mostrar la cookie

let cookies = document.cookie.split("; ");

let nombreCookie = cookies.find(c => c.startsWith("nombreUsuario="));

let nombreUsuario = nombreCookie.split("=")[1];

alert(`¡Bienvenido de nuevo, ${nombreUsuario}!`);

}

### **La función setCookie()**

La función setCookie() no es nativa de JavaScript, pero se puede crear una para manejar fácilmente la creación y configuración de cookies en el navegador. Esta función es útil para encapsular la lógica necesaria y aceptar parámetros que definan las propiedades de la cookie.

### **Ejemplo de función** setCookie()

Aquí tienes un ejemplo de una implementación básica de setCookie():

function setCookie(nombre, valor, dias, path = "/") {

let cookie = `${nombre}=${encodeURIComponent(valor)}`;

if (dias) {

const fecha = new Date();

fecha.setTime(fecha.getTime() + dias \* 24 \* 60 \* 60 \* 1000); // Convertir días a milisegundos

cookie += `; expires=${fecha.toUTCString()}`;

}

cookie += `; path=${path}`; // Definir el path de la cookie

document.cookie = cookie;

}

### **Parámetros explicados**

1. nombre: Nombre de la cookie.
2. valor: Valor de la cookie. Se codifica con encodeURIComponent para manejar caracteres especiales.
3. dias: Tiempo de vida de la cookie en días. Si es null o no se define, la cookie será una cookie de sesión, que se eliminará cuando el navegador se cierre.
4. path: Especifica el alcance de la cookie dentro del dominio. Por defecto, se establece en la raíz (/).

### **Uso básico de** setCookie()

1. Crear una cookie con tiempo de vida de 7 días

setCookie("usuario", "Juan", 7);

console.log(document.cookie); // "usuario=Juan"

1. Crear una cookie de sesión (sin tiempo de expiración)

setCookie("tema", "oscuro");

console.log(document.cookie); // "usuario=Juan; tema=oscuro"

1. Definir el path de la cookie

setCookie("idioma", "es", 7, "/miApp");

console.log(document.cookie); // Solo accesible en el path "/miApp"

### **Extender la función** setCookie() **con más opciones**

Podemos agregar soporte para más atributos como secure, SameSite o domain:

function setCookie(nombre, valor, dias, opciones = {}) {

let cookie = `${nombre}=${encodeURIComponent(valor)}`;

if (dias) {

const fecha = new Date();

fecha.setTime(fecha.getTime() + dias \* 24 \* 60 \* 60 \* 1000);

cookie += `; expires=${fecha.toUTCString()}`;

}

// Opciones adicionales

if (opciones.path) cookie += `; path=${opciones.path}`;

if (opciones.domain) cookie += `; domain=${opciones.domain}`;

if (opciones.secure) cookie += `; secure`;

if (opciones.sameSite) cookie += `; SameSite=${opciones.sameSite}`;

document.cookie = cookie;

}

### **Uso de la función extendida**

1. Cookie con múltiples opciones

setCookie("token", "abc123", 3, {

path: "/",

secure: true,

sameSite: "Strict",

});

console.log(document.cookie); // "token=abc123"

1. Cookie válida para un subdominio

setCookie("session", "active", 1, {

domain: "sub.miSitio.com",

path: "/",

});

### **Notas importantes**

1. Codificación y decodificación de valores:
   * Los valores deben codificarse con **encodeURIComponent()** y decodificarse con **decodeURIComponent()** para manejar caracteres especiales.
   * Ejemplo:

setCookie("user", "Juan Pérez");

console.log(document.cookie); // "user=Juan%20P%C3%A9rez"

1. Borrar cookies: Para eliminar una cookie, usa un valor vacío y una fecha de expiración pasada:

setCookie("usuario", "", -1);

console.log(document.cookie); // Cookie eliminada

1. Limitaciones del navegador:
   * Las cookies tienen un límite de tamaño (generalmente 4 KB por cookie).
   * Cada dominio tiene un límite de cantidad de cookies que puede almacenar (generalmente 20-50).

### **La función getCookie()**

La función getCookie() es una herramienta útil para recuperar el valor de una cookie almacenada en el navegador a partir de su nombre.

### **Cómo funciona** document.cookie

* document.cookie devuelve todas las cookies disponibles para el sitio web actual en forma de un único string.
* Cada cookie tiene el formato: nombre=valor, y las cookies están separadas por ; .

Ejemplo:

document.cookie = "usuario=Juan";

document.cookie = "tema=oscuro";

console.log(document.cookie);

// "usuario=Juan; tema=oscuro"

Por lo tanto, para obtener una cookie específica, debemos buscar su nombre y extraer su valor.

### **Implementación básica de** getCookie()

Aquí tienes una implementación básica de la función:

function getCookie(nombre) {

const cookies = document.cookie.split("; "); // Separar las cookies

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

const [key, value] = cookies[i].split("="); // Separar nombre y valor

if (key === nombre) {

return decodeURIComponent(value); // Devolver el valor decodificado

}

}

return null; // Devolver null si la cookie no existe

}

### **Uso básico de** getCookie()

1. Crear algunas cookies:

document.cookie = "usuario=Juan";

document.cookie = "tema=oscuro";

1. Recuperar el valor de una cookie:

console.log(getCookie("usuario")); // "Juan"

console.log(getCookie("tema")); // "oscuro"

console.log(getCookie("inexistente")); // null (no existe)

### **Mejorando** getCookie()

La versión anterior recorre todas las cookies una a una. Si quieres optimizar el proceso o manejar cookies con espacios o caracteres especiales, puedes hacerlo con expresiones regulares:

function getCookie(nombre) {

const match = document.cookie.match(new RegExp(`(?:^|; )${nombre}=([^;]\*)`));

return match ? decodeURIComponent(match[1]) : null;

}

Explicación de la expresión regular:

1. (?:^|; ): Busca el inicio de una cookie o un separador ; .
2. ${nombre}: Busca el nombre de la cookie.
3. =([^;]\*): Captura el valor de la cookie (todo lo que hay después del = hasta el siguiente ; o el final).
4. match[1]: Recupera el valor capturado por la expresión regular.

### **Ejemplo de uso avanzado**

1. Establecer cookies:

document.cookie = "usuario=Juan";

document.cookie = "tema=oscuro";

document.cookie = "idioma=es";

1. Recuperar cookies:

console.log(getCookie("usuario")); // "Juan"

console.log(getCookie("idioma")); // "es"

console.log(getCookie("inexistente")); // null

### **Notas importantes**

1. Espacios y caracteres especiales:
   * Los valores de cookies deben codificarse con encodeURIComponent() al establecerlos y decodificarse con decodeURIComponent() al leerlos.
   * Ejemplo:

document.cookie = "nombre=Juan%20P%C3%A9rez";

console.log(getCookie("nombre")); // "Juan Pérez"

1. Cookies de sesión y alcance:
   * Si una cookie está configurada con un path o domain que no coincide con la página actual, document.cookie no podrá acceder a ella.
   * Ejemplo:

document.cookie = "test=1; path=/miApp";

console.log(getCookie("test")); // null (si no estás en "/miApp")

1. Cookies seguras:
   * Las cookies configuradas con el atributo Secure solo están disponibles a través de conexiones HTTPS.

### **Eliminar cookies**

Eliminar cookies en el navegador implica establecer la cookie con una fecha de expiración pasada, ya que el navegador automáticamente elimina las cookies expiradas.

### **Cómo funcionan las cookies y su expiración**

* Cada cookie tiene una propiedad expires o max-age que determina su duración.
* Si expires tiene una fecha pasada o max-age es 0, el navegador elimina la cookie.

### **Métodos para borrar cookies**

#### 1. Establecer la cookie con una fecha de expiración pasada

Este es el método más común. Se reescribe la cookie con el mismo nombre y expires configurado a una fecha anterior.

Ejemplo:

document.cookie = "usuario=Juan; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

### **Función para borrar cookies**

#### Implementación básica

function deleteCookie(nombre, path = "/") {

document.cookie = `${nombre}=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=${path}`;

}

#### Uso:

// Crear y luego eliminar una cookie

document.cookie = "usuario=Juan; path=/";

console.log(document.cookie); // "usuario=Juan"

deleteCookie("usuario");

console.log(document.cookie); // ""

### **Notas importantes al eliminar cookies**

1. Path y domain deben coincidir:
   * Para borrar una cookie, el path y el domain deben coincidir con los que se usaron al crearla.
   * Ejemplo:

// Crear cookie con path específico

document.cookie = "test=123; path=/miApp";

// Intentar borrar sin path correcto (fallará)

deleteCookie("test");

console.log(document.cookie); // "test=123"

// Borrar con el path correcto

deleteCookie("test", "/miApp");

console.log(document.cookie); // ""

1. Cookies con atributos Secure y HttpOnly:
   * Las cookies con el atributo Secure solo pueden ser manejadas en HTTPS.
   * Las cookies con HttpOnly no son accesibles desde JavaScript.

### **Eliminar todas las cookies**

Aunque no existe un método directo para borrar todas las cookies, puedes hacerlo listando las cookies disponibles y eliminándolas una por una.

#### Ejemplo de función:

#### Uso:

// Crear varias cookies

document.cookie = "usuario=Juan; path=/";

document.cookie = "tema=oscuro; path=/";

document.cookie = "idioma=es; path=/";

// Eliminar todas las cookies

deleteAllCookies();

console.log(document.cookie); // ""

### **Extensión para manejar** domain **y** Secure

Si necesitas eliminar cookies que fueron creadas con un dominio o protocolo específico, puedes agregar opciones adicionales a las funciones de borrado:

function deleteCookie(nombre, path = "/", domain = null) {

let cookie = `${nombre}=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=${path}`;

if (domain) {

cookie += `; domain=${domain}`;

}

document.cookie = cookie;

}

### **Conclusión**

Borrar cookies es una operación simple cuando entiendes los atributos como path, domain y Secure. Si necesitas borrar cookies en aplicaciones más complejas, herramientas como [js-cookie](https://github.com/js-cookie/js-cookie) facilitan esta tarea.