

# Web Storage

## Web storage

- ✗ Este API permite guardar información de forma local, es decir, en la parte del cliente.
- ✗ Es más eficiente que las cookies. Permite almacenar cantidades de datos mucho mayores, siempre en modo texto.
- ✗ Al contrario que las cookies, la información almacenada mediante este API **SÓLO** es accesible desde el navegador, **NO** desde el servidor.

<u>Comparativa Cookies / Web Storage</u>	COOKIES	WEB STORAGE
Capacidad de almacenamiento	4KB aprox.	5MB a 10MB, aprox.
Peticiones	Recurrencia	Mejor rendimiento
Acceso	getCookie(), setCookie()	clave/valor

## Web storage

### Tipos de almacenamiento

Este API proporciona dos tipos de almacenamiento:

- ✗ **sessionStorage**. Este sistema de **almacenamiento** guarda los datos de forma **temporal** durante la sesión de navegación en una ventana del navegador. Es decir, sólo estarán disponibles en la ventana del navegador en la que fueron guardados y hasta que ésta se cierre.
- ✗ **localStorage**. Este sistema de **almacenamiento** permite guardar los datos de forma **permanente** siendo, además, accesibles desde todas las ventanas del navegador, no sólo desde la ventana en la que fueron guardados. Estarán disponibles hasta que el usuario los elimine.

## Web storage

### Comprobación de que el navegador soporta el API

Para comprobar si el navegador soporta el API Web Storage basta con el siguiente código JavaScript:

```
if(typeof(Storage)!="undefined")
{
    // También es posible utilizar:
    //     if(window.localStorage) ...
    //     if(window.sessionStorage) ...
    // localStorage y sessionStorage soportados
    // Aquí va el código.....
}
else
{
    // No hay soporte para el API Web Storage
    alert("Tu navegador no soporta Web Storage")
}
```

## Web storage

### Interfaz Storage

Tanto **sessionStorage** como **localStorage** implementan la interfaz **Storage**.

```
Interfaz Storage{
  readonly attribute unsigned long length;
  DOMString? key(unsigned long index);
  getter DOMString getItem(DOMString key);
  setter creator void setItem(DOMString key, DOMString value);
  deleter void removeItem(DOMString key);
  void clear();
};
```

- **length**: Devuelve el número de pares clave/valor.
- **key**(*n*): Devuelve el nombre de la clave que ocupa la posición *n*, o *null* si no existe.
- **getItem**(*key*): Devuelve el valor asociado a la clave *key*.
- **setItem**(*key*, *value*): Si la clave *key* no existe, crea un nuevo par clave/valor con *key/value*. Si la clave ya existe, actualiza su valor a *value*.
- **removeItem**(*key*). Borra el par clave/valor cuya clave coincida con *key*.
- **clear**(): Borra de la lista todos los pares clave/valor.

## Web storage

### Ejemplo:

#### Código JavaScript

```
function comprobar(){
    if(window.localStorage){ // Se comprueba si hay soporte para Web Storage
        var frm = document.querySelectorAll("form")[0];
        if(frm.ckbGuardar.checked){ // Si se ha marcado guardar datos ...
            localStorage.setItem("login", frm.login.value);
            localStorage["pass"] = frm.pass.value; // modo alternativo
        }
    }
}

function rellenar(){ // Se comprueba si hay soporte para Web Storage
    if(window.localStorage){
        var frm = document.querySelectorAll("form")[0];
        if(localStorage.getItem("login")){ // Si hay datos en loginStorage ...
            frm.login.value = localStorage.getItem("login");
            frm.pass.value = localStorage.pass; // modo alternativo
        }
    }
}
```

#### Código HTML

```
<body onload="rellenar();">
  <form onsubmit="return comprobar();">
    Login:<input type="text" name="login"><br>
    Password:<input type="password" name="pass"><br>
    <input type="checkbox" name="ckbGuardar"> Usar localStorage<br>
    <input type="submit" value="Enviar">
  </form>
</body>
```