

Almacenamiento local

UD1: Tecnologías para el desarrollo de interfaces



Al acabar la lección..

- Sabrás lo que es una cookie
 - Habrás aprendido a crear, modificar, eliminar y consultar una cookie con JavaScript
 - Habrás aplicado lo aprendido a un caso “real”
- Conocerás la alternativa que ofrece HTML5 para las Cookies, y comprobarás sus ventajas
 - Probarás los principales métodos
 - Aplicarás lo aprendido a un caso “real”



Indice

- Cookies
- Almacenamiento local
- JSON



Cookies

[Indice](#)



Cookies

- Son **ficheros de texto** que se almacenan en determinada carpeta en el navegador
 - Chrome
 - [user] \ Local Settings \ Application Data \ Google \ Chrome \ User Data \ Default.
 - Mozilla. En el perfil del navegador.
 - [windows]\Application Data\Mozilla\Profiles\[profilename]\;
 - IE: Archivos temporales de Internet.
- A grandes rasgos:

```
todasLasCookies = document.cookie; //Mostrar todas las cookies  
document.cookie = nuevaCookie; //Escribir una cookie
```

Cookies: crear una cookie

Si no existen una cookie con el nombre la creará automáticamente.
Si ya existe una cookie con el nombre se reemplaza.

OBLIGATORIO

```
document.cookie = "nombreCookie=datosCookie"
```

OPCIONAL

```
[; expires=horaformatoGMT]  
[; path=ruta]  
[; domain=nombreDominio]  
[; secure] "
```

```
document.cookie="username=manolo";  
document.cookie="genero=masculino";
```

Grabar una cookie: Otros parámetros

Parámetro	Significado
expires	Fecha de caducidad expires=Thu, 01-Jan-15 00:00:01 GMT; Si no se establece la cookie dura el tiempo de la sesión (hasta cierre de navegador).
path	Ruta de la Web
domain	Si no se pone pertenece al dominio de la página que la creó
secure	Nuestra cookie será accesible por cualquier programa en el dominio.

```
document.cookie="username=manolo; expires=Thu,  
16 Nov 2025 12:00:00 UTC;"
```

Cookies: Recuperar una cookie

- Accedemos a **document.cookie**
 - Debemos obtener la cadena de texto de la misma y extraer los datos necesarios de su contenido.

```
function getCookie(nomCookie) {  
    var cook=document.cookie.split(";"); // pares de valores  
  
    for (var i=0; i<cook.length; i++) { // revisamos todos los pares  
        var n = cook[i].split("="); // separamos nombre/valor  
        var nombre=n[0];  
        var valor =n[1];  
        if (nombre.trim()==nomCookie.trim()) // si es el buscado  
            return valor;// devolvemos su valor  
    }  
    return null; // si no se encuentra = nulo  
}
```


Cookies: Modificar una cookie

- Se le asigna un nuevo valor usando el mismo nombre que el crearla
 - IMPORTANTE: Hay que repetir todos los parámetros que en su creación, sino se crea una cookie nueva

```
document.cookie="genero=femenino";
```

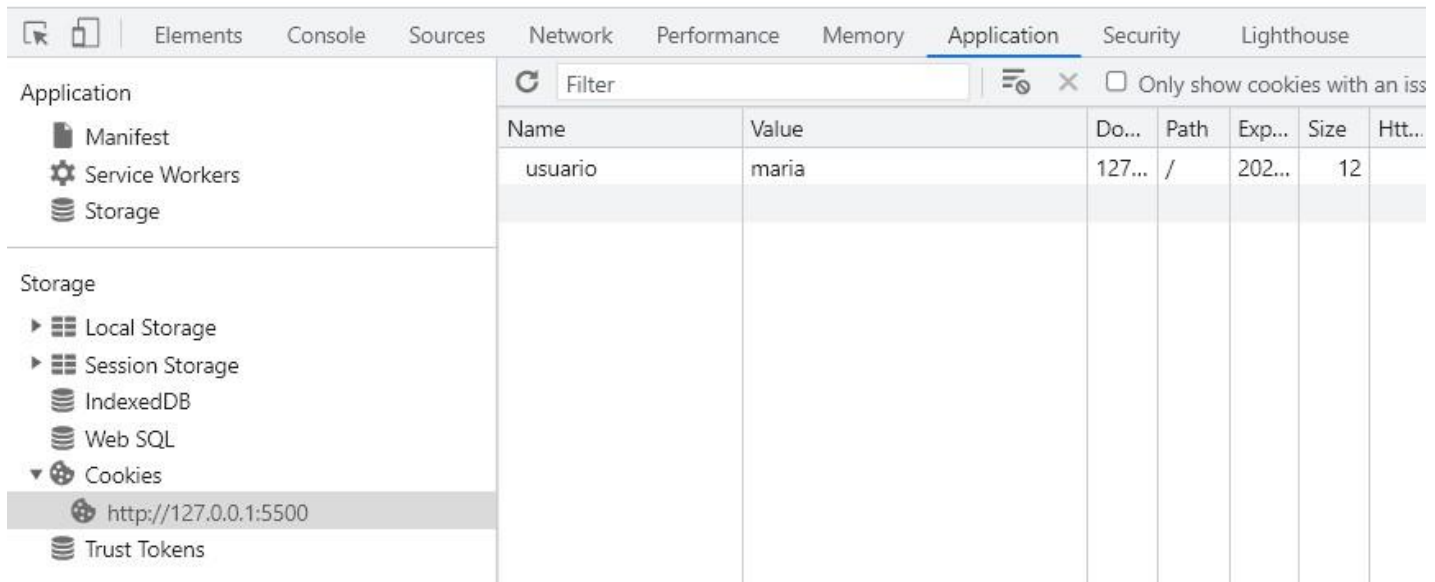
Cookies: borrar una cookie

- Se fija su fecha de expiración a un momento pasado:

```
document.cookie="username=; expires=Thu, 01 Jan 1970  
00:00:01 GMT;";  
document.cookie="genero=; expires=Thu, 01 Jan 1970  
00:00:01 GMT; ";
```

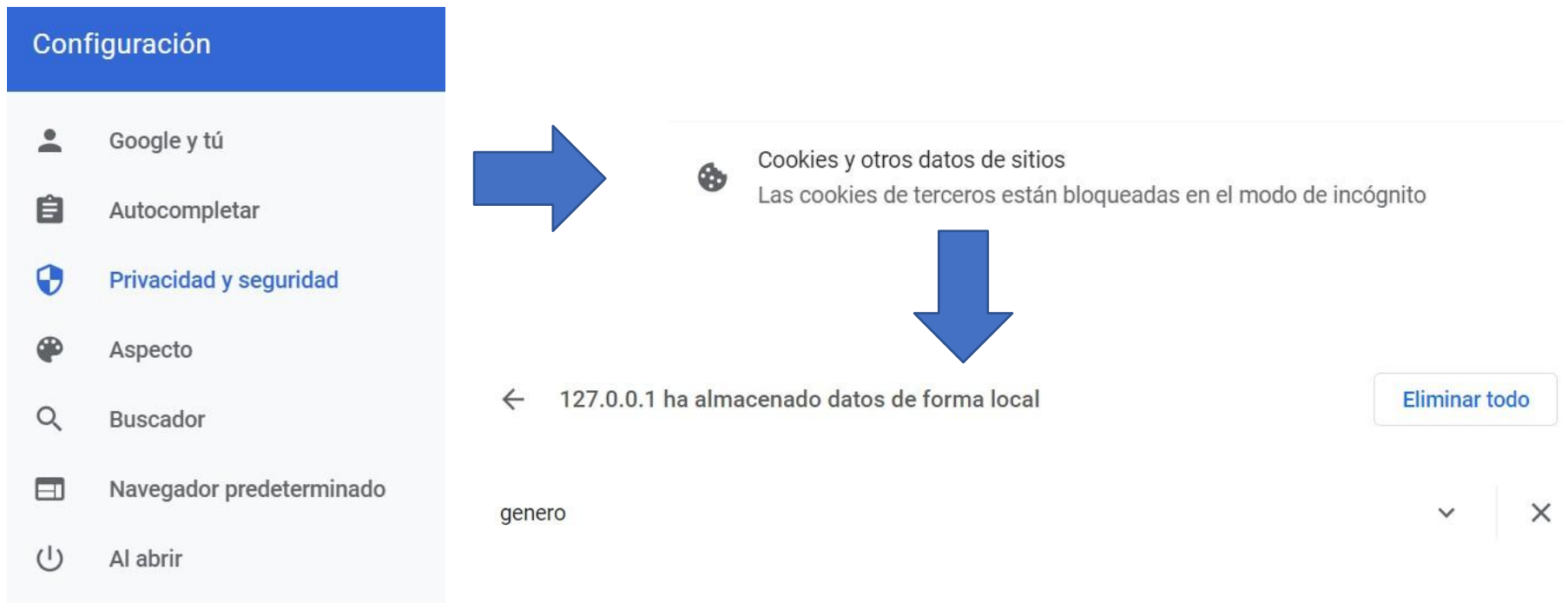
En la consola...

- En Chrome, podemos ver las Cookies en la sección **Application** de la consola:



En nuestro navegador...

- En la sección de configuración de nuestro navegador podemos gestionar las cookies. Por ejemplo, en Chrome:



Código útil con Cookies

```
function setCookie(nombre, valor, caducidad) {  
    var hoy = new Date();  
    var caduca = new Date(now.getTime() + caducidad);  
  
    document.cookie = nombre + "=" + valor + "; expires=" +  
        caduca.toGMTString() + "; path=/";  
}
```

```
function haycookies(){  
    document.cookie="micookie=hay";  
    return ( getCookie("micookie") == "hay" );  
}
```

Cookies: Limitaciones

- La cookie viaja en las cabeceras de cada petición HTTP.
- Tamaño limitado (4KB) → Permite guardar muy poca información
 - Habitualmente un identificador de sesión que sirve para recuperar información adicional del servidor.



Almacenamiento local

[Indice](#)

Almacenamiento local (API HTML5)

- HTML5 dispone de 3 tecnologías que permiten que las aplicaciones almacenen datos en el cliente:
 - **WebStorage**. Permite almacenar parejas de clave/valor
 - **Web SQL Database**. Sistema de almacenamiento basado en SQL
 - No va a ser mantenido en el futuro, aunque su uso está extendido.
 - **IndexedDB**. Sistema de almacenamiento basado en objetos.
 - Para almacenar cantidades de datos mayores
 - API más complicada

WebStorage

- Permite almacenar una cantidad de datos mucho mayor que las cookies (hasta 5-10M MB)
 - Más potente.
- No tiene caducidad
- No transmite los datos en cada petición HTTP
 - Mejor rendimiento
 - Mejor seguridad.

localStorage y sessionStorage

- Son dos implementaciones de la interfaz **Storage** que permiten el almacenamiento local.
 - **sessionStorage** actúa sobre el ámbito de la sesión de la ventana o pestaña.
 - **localStorage** permite que los datos perduren indefinidamente.

localStorage y sessionStorage son objetos distintos e independientes

localStorage y sessionStorage

- Permiten realizar operaciones de almacenamiento y recuperación de **cadenas**.

Método	Significado
<code>getItem(key)</code>	Devuelve el valor correspondiente a la clave <code>key</code> .
<code>setItem(key,value)</code>	Almacena el <code>value</code> referenciado por la cadena <code>key</code> .
<code>removeItem(key)</code>	Elimina el par clave/valor con clave igual a <code>key</code> .
<code>length</code>	Atributo que contiene el número de elementos par/valor almacenados.
<code>key(i)</code>	Devuelve la clave del elemento que está en la posición <code>i</code> .
<code>clear()</code>	Elimina todos los elementos.

Comprobando la compatibilidad

- Práctica conveniente pues versiones obsoletas no lo permiten (debemos buscar una alternativa)

```
function compruebaCompatibilidad(){  
    if(typeof(Storage) !== "undefined")  
        console.info("Tu navegador acepta almacenamiento local");  
    else  
        console.info("Tu navegador NO acepta almacenamiento local");  
}
```

Guardando datos

- La clave identifica de manera unívoca el valor que guardo.

```
function guardaDatos(clave, valor){  
    localStorage.setItem(clave,valor);  
}
```

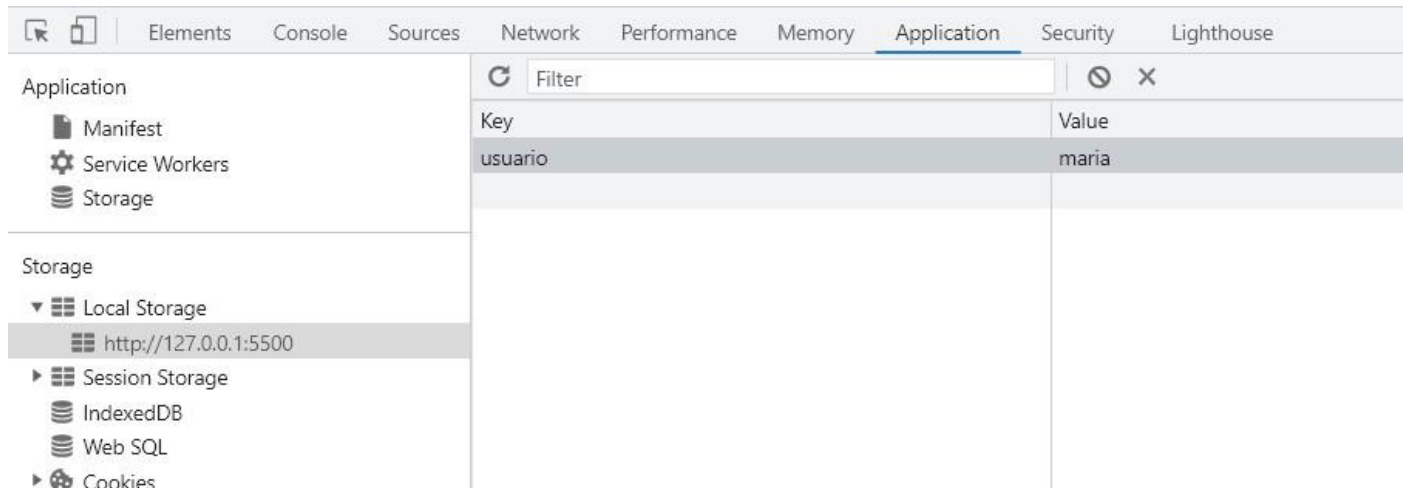
```
guardaDatos("jugador1","Fermín");  
guardaDatos("jugador1","Pachín");
```

- También podríamos usar esta sintaxis

```
localStorage.jugador1="Fermín";
```

En el navegador...

- En Chrome, podemos ver el contenido de **localStorage** y **sessionStorage** en la sección **Application**:



Recuperando datos

- El valor se recupera sabiendo la clave usando la función **getItem**:

```
function recuperaDatos(clave){  
    return localStorage.getItem(clave);  
}
```

```
info=document.getElementById("info");  
info.innerHTML=recuperaDatos("jugador1");
```

- Si queremos recuperar TODOS los datos:

```
function muestraTodosLosDatos(){  
    var info=document.getElementById("info");  
    info.innerHTML="";  
    for(var i=0; i<localStorage.length;i++){  
        var clave=localStorage.key(i);  
        var contenido=localStorage.getItem(clave);  
        info.innerHTML+=clave+": "+contenido+"<br/>"  
    }  
}
```

localStorage.key(i)
Devuelve la clave del
elemento que está en la
posición i.

Borrando datos

- El valor se borra sabiendo la clave usando la función **removeItem**:

```
function eliminar(clave){  
  localStorage.removeItem(clave);  
}
```

- Si queremos borrar TODOS los datos:

```
localStorage.clear();
```




JSON

[Indice](#)

Almacenando objetos

- En ocasiones nos puede interesar almacenar objetos en lugar de un único String
 - Podemos hacer uso de JSON



JSON (*JavaScript nOtatioN*)



- Es un formato de intercambio de datos que sigue una sintaxis muy parecida al modo en que almacenamos objetos en JavaScript.
- Intenta reemplazar a XML, pues es mucho más sencillo de procesar.
- Es aceptado de manera nativa por JavaScript

```
MovieList.json
{
  "result": [
    {
      "actor": "Vivien Leigh",
      "title": "Gone with the Wind",
      "director": "Victor Fleming",
      "description": "Going with the wind"
    },
    {
      "actor": "Michael J Fox",
      "title": "Back To The Future",
      "director": "Robert Zemeckis",
      "description": "Going back to the future"
    }
  ]
}
```

Serializando objetos con JSON

- ***Serializar:*** Convertir un objeto a una representación que puedo almacenar fácilmente de manera persistente
- Entre otros, JSON proporciona un par de métodos que permiten representar un objeto como cadena y viceversa:

<code>JSON.stringify(objeto)</code>	Devuelve una representación como String del objeto que le paso.
<code>JSON.parse(cadena)</code>	Devuelve un objeto de acorde con la representación en cadena que le paso.

Convirtiendo a JSON: Ejemplo

```
let jugador={  
  nombre:"Godzilla",  
  porcentajeVida:100,  
  nivel:3,  
  tiempo:"1:00",  
  trucos:false,  
}
```

```
var jugadorJSON=JSON.stringify(jugador);  
console.log(jugadorJSON);
```

```
{"nombre":"Godzilla","porcentajeVida":100,"nivel":3,"tiempo":"1:00","trucos":false}
```

Combinado con Web Storage

```
var jugador1=new Jugador("Scorpion",100,1,"2:00",false);  
var jugador2=new Jugador("Sub-Zero",100,2,"1:30",true);  
var jugador3=new Jugador("Rayden",100,3,"2:00",false);
```

```
sessionStorage.setItem("jugador"+jugador1.id,JSON.stringify(jugador1));  
sessionStorage.setItem("jugador"+jugador2.id,JSON.stringify(jugador2));  
sessionStorage.setItem("jugador"+jugador3.id,JSON.stringify(jugador3));
```

Elements	Resources	Network	Sources	Timeline	Profiles	Audits	Console
▼ Frames				Key	Value		
▼ (storage.html)				jugador0	{\"id\":0,\"nombre\":\"Scorpion\",\"porcentajeVida\":100,\"nivel\":1,\"tiempo\":\"2:00\"}		
▶ Scripts				jugador1	{\"id\":1,\"nombre\":\"Sub-Zero\",\"porcentajeVida\":100,\"nivel\":2,\"tiempo\":\"1:30\"}		
▶ Stylesheets				jugador2	{\"id\":2,\"nombre\":\"Rayden\",\"porcentajeVida\":100,\"nivel\":3,\"tiempo\":\"1:00\"}		

Para recuperar
los datos...

```
for(var i=0;i<sessionStorage.length;i++){  
    var jugadorJSON=sessionStorage.getItem("jugador"+i);  
    var jugador=JSON.parse(jugadorJSON);  
    alert(jugador.nombre);  
}
```