



## Parte 4: Almacenamiento local

UD3: Interacción con el usuario y modelo de objetos del documento

María Rodríguez Fernández [mariarfer@educastur.org](mailto:mariarfer@educastur.org)

# Al final de este documento...



- 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”

# 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.
  - Etc.
- A grandes rasgos:

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

# Cookies: crear una cookie

- Sintaxis ([ ] → Opcional)

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

**Obligatorio  
aunque sea  
vacío**

```
document.cookie = "nombreCookie=datosCookie  
[; expires=horaformatoGMT]  
[; max-age=duracion]  
[; path=ruta]  
[; domain=nombreDominio]  
[; secure] "
```

– **Ejemplo:**

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

# Grabar una cookie: Otros parámetros

- 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).
- max-age. Duración máxima en segundos
- path. Ruta actual de nuestra Web
- domain. Si no se pone es el dominio de la página que creó la cookie
- secure. Nuestra cookie será accesible por cualquier programa en el dominio
  - **Ejemplo:**

```
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 y borrar una cookie

- **Modificar:** 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

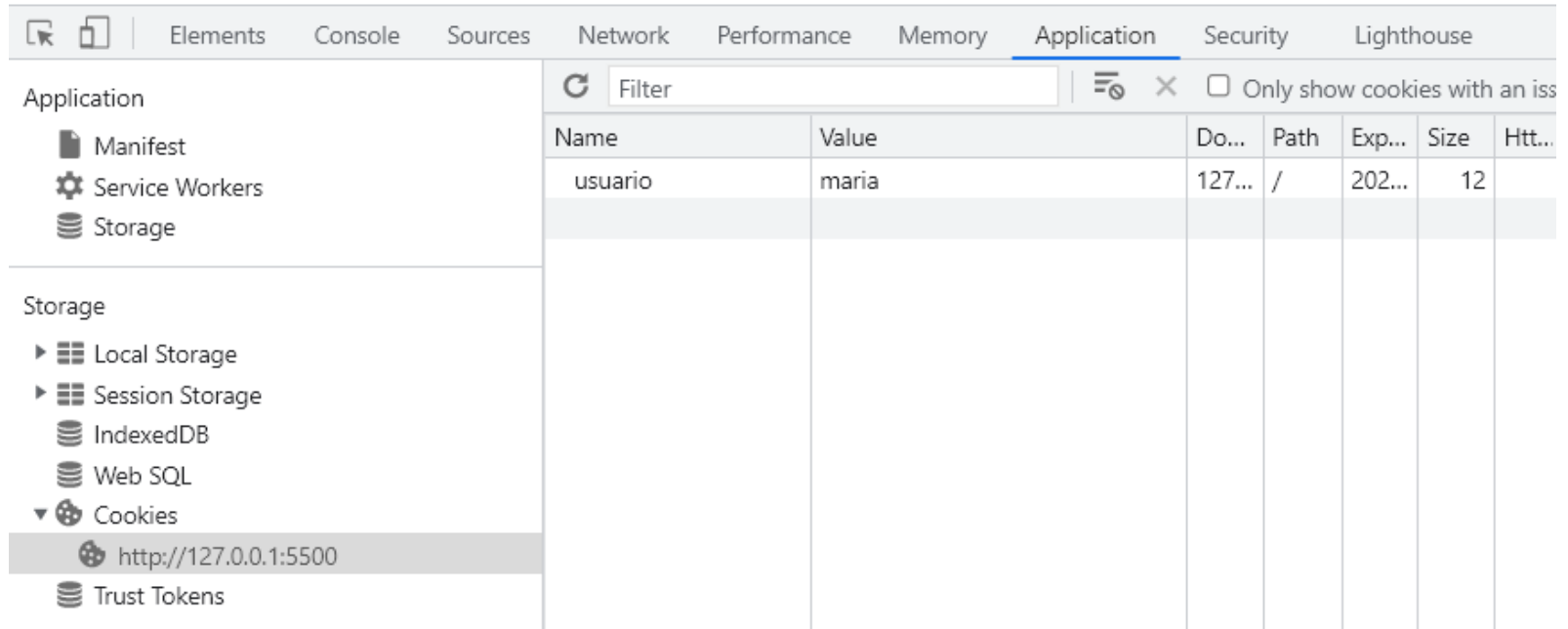
```
document.cookie="genero=femenino";
```

- **Borrar:** 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:



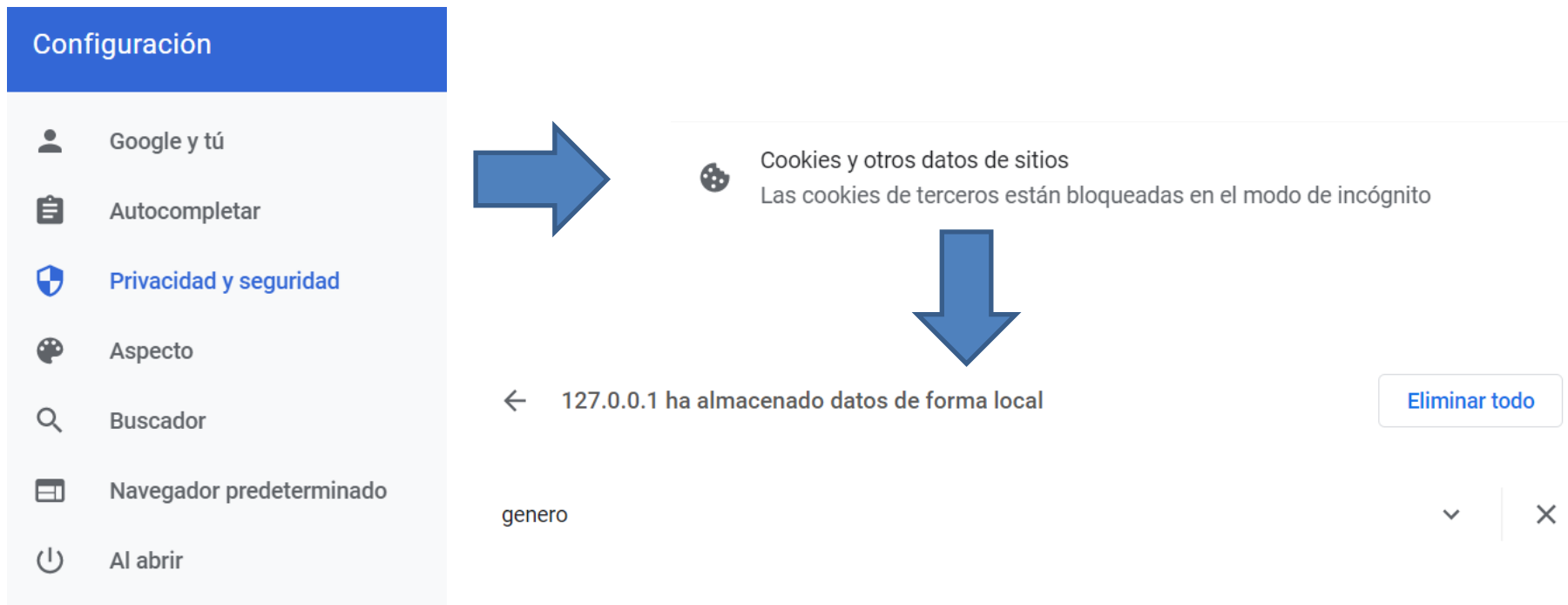
The screenshot shows the Chrome DevTools 'Application' tab. On the left sidebar, under 'Storage', the 'Cookies' folder is expanded, showing a list of cookies for the URL 'http://127.0.0.1:5500'. The main panel displays a table of cookies.

Name	Value	Do...	Path	Exp...	Size	Htt...
usuario	maria	127...	/	202...	12	



# En nuestro navegador...

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



# Códigos útiles con Cookies

- Función crear Cookie (para no repetir el mismo código):

```
function setCookie(nombre, valor, caduca) {  
    var hoy = new Date();  
    hoy.setTime(hoy.getTime()+caduca);  
    var expiracion = "expires="+d.toUTCString();  
    document.cookie = nombre+"="+valor+";"+expiracion+";path="/";  
}
```

- Antes de trabajar con cookies podemos comprobar si están activas en el navegador
  - Ej: Podemos intentar guardar un valor y luego recuperarlo

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

# EJERCICIO PROPUESTO V (1 de 2)

- A partir del siguiente código HTML, implementa la funcionalidad de cada botón creando una función para cada manejador

Ver todas las cookies

Crear cookie

Modificar cookie

Leer cookie

Borrar cookie

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Cookies - Ejemplo completo</title>
</head>
<body>
  <button type="button" id="verTodas">Ver todas las cookies</button>
  <button type="button" id="crearCookie">Crear cookie </button>
  <button type="button" id="modificarCookie">Modificar cookie </button>
  <button type="button" id="leerCookie">Leer cookie </button>
  <button type="button" id="borrarCookie">Borrar cookie </button>
</body>
</html>
```

# EJERCICIO PROPUESTO V (2 de 2)

- **verCookies:** accede a **document.cookie** y lo muestra. Se le llamará también después de cada operación realizada.
- **crearModifCookie:** pide nombre, valor y días para expirar con prompt y llama a **setCookie** para crearla:
  - **setCookie (nombre, valor, fecha):** crea una nueva cookie con los valores que se le pasan, calculando la fecha de expiración a partir de la actual, añadiéndole los segundos correspondientes (*ver transparencia 11*)
- **borrarCookie:** pide el nombre y llama a **deleteCookie:**
  - **deleteCookie (nombre):** borra la cookie poniendo su fecha a 0
- **leerCookie:** pide el nombre de la cookie y llama a **getCookie:**
  - **getCookie (nombre):** recorre todas las cookies de document.cookie y compara el nombre con el buscado. Si lo encuentra devuelve el valor.

¡OJO! Es necesario que ejecutes el código en un servidor (ej. Go Live de VS Code)

# EJERCICIO PROPUESTO IV

## (Continuación 1 de 2)



- Modifica el formulario para que las comprobaciones se hagan al pulsar el botón "Enviar Datos"
  - `checkValidity` al hacer click en el botón
  - Evento `invalid` en cada campo para mostrar el error
  - Evento `input` para limpiar el error
- Lleva cuenta de los errores que hay usando un gestor de errores
  - Ayuda al usuario a ir arreglando uno a uno

Nombre

Apellidos

Fecha de nacimiento

DNI

Email

Contraseña

Contraseña (repetición)

Género ☒ Hombre ☐ Mujer

☐ Suscribirme al boletín de novedades

☐ Informarme sobre ofertas

Producto favorito

Comentario

Este es tu primer intento

# EJERCICIO PROPUESTO IV

## (Continuación 2 de 2)



- Mejora el formulario del ejercicio propuesto V para que:
  - Guarde el número de intentos en una cookie llamada "contador"
  - Haz que no se pueda mandar el formulario cuando se supere un número máximo de intentos

Nombre	<input type="text"/>
Apellidos	<input type="text"/>
Fecha de nacimiento	<input type="text" value="dd/mm/aaaa"/>
DNI	<input type="text"/>
Email	<input type="text"/>
Contraseña	<input type="password"/>
Contraseña (repetición)	<input type="password"/>
Género	<input checked="" type="radio"/> Hombre <input type="radio"/> Mujer
	<input type="checkbox"/> Suscribirme al boletín de novedades
	<input type="checkbox"/> Informarme sobre ofertas
Producto favorito	<input type="text" value="Elige uno..."/>
Comentario	<input type="text" value="Introduce tu comentario..."/>
<input type="button" value="Enviar datos"/>	
<div>Este es tu primer intento</div>	

# 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 (API HTML5)

- HTML5 dispone de 3 tecnologías que permiten que las aplicaciones almacenen datos en el cliente:
  - **WebStorage**. Permite almacenar parejas de clave/valor
  - **IndexedDB**. Sistema de almacenamiento basado en objetos.
    - Para almacenar cantidades de datos mayores
    - API más complicada
- Tendencias:
  - **API Caché**: Sigue la idea de almacenar partes del sitio web para que luego estén disponibles sin conexión
    - <https://developer.mozilla.org/en-US/docs/Web/API/Cache>



# WebStorage

- Características:
  - 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
- 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

- Permiten realizar operaciones de almacenamiento y recuperación de **cadena**s.

<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);  
}
```

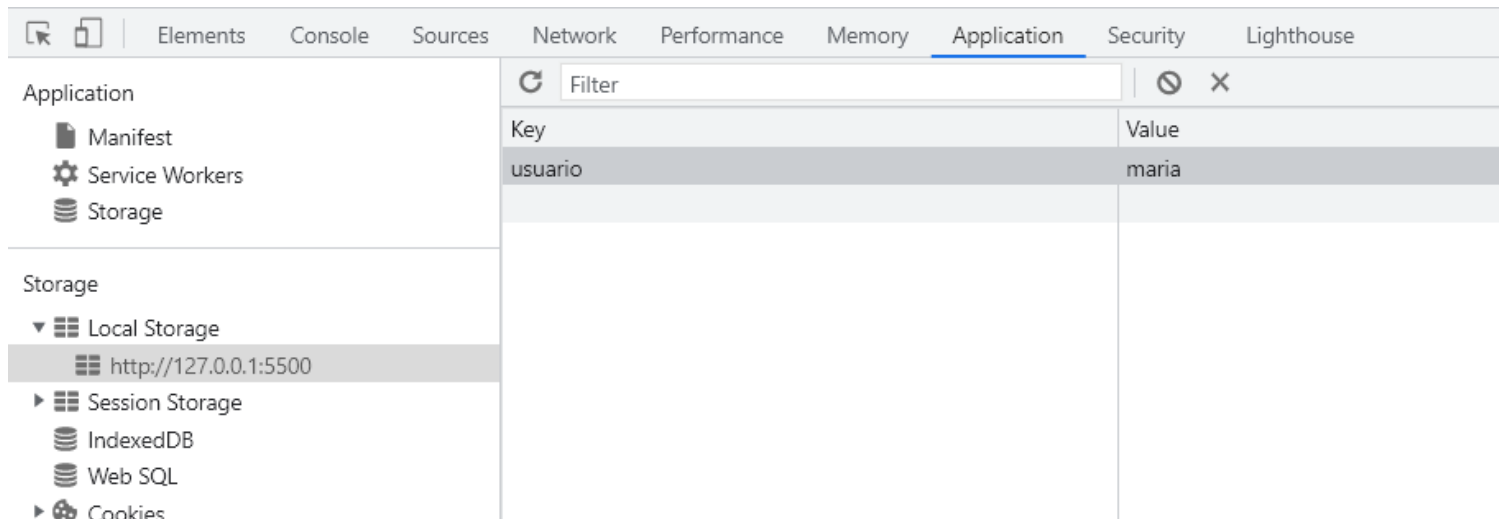
```
guardaDatos("jugador1","Fermín");  
guardaDatos("jugador1","Pachín");
```

- También podríamos usar esta sintaxis pero es menos recomendable:

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

# En el navegador...

- Por ejemplo 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();
```

**localStorage** y **sessionStorage**  
son objetos distintos e  
independientes

# EJERCICIO PROPUESTO VI (1 de 3)

- A partir del siguiente código HTML:

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>WebStorage - Ejemplo completo</title>
</head>
<body>
  <button type="button" id="incrementar">Incrementar</button>
  <button type="button" id="decrementar">Decrementar</button>
  <button type="button" id="logout">Log out</button>
  <p id="saludo"></p>
  <p id="contador"></p>
</body>
</html>
```

Incrementar

Decrementar

Log out





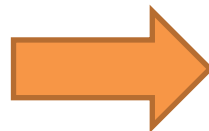
# EJERCICIO PROPUESTO VI (2 de 3)

- Comprobar **si el navegador soporta Storage** y mostrar un mensaje con el nombre del usuario si este se ha logueado en la página, diferenciando si es su primera visita o ya ha entrado en más ocasiones.
- Utilizaremos una función para llevar el control de las **veces** que el usuario ha accedido a la página.
- Crearemos dos funciones para **incrementar** y **decrementar** el valor del contador anterior.
- Finalmente crearemos otra función para hacer **logout** y **restablecer** los valores anteriores.

Incrementar Decrementar Log out

¡Tu primera visita, maria!

Contador: 0



Incrementar Decrementar Log out

¡Bienvenido/a de nuevo, maria!

Contador: 6

# EJERCICIO PROPUESTO VI (3 de 3)

- Comprueba:
  - Si cierras la ventana recuerda tu nombre
  - Si haces logout deja de recordar tu nombre
  - El contador es válido en cada sesión:
    - Si refrescas la página se mantiene
    - Si cierras la ventana se pierde

# EJERCICIO PROPUESTO IV

## (Continuación)



- Guarda los datos del formulario en **localStorage**
- Carga los datos desde **localStorage** (si existen) al cargar la página
- RETO: Utiliza JSON
  - Consulta los apuntes EXTRA sobre JSON para guardar un objeto de la clase Contacto directamente

