

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

## **Almacenamiento Web**

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

de Administración Pública  
NURIA ARAGON MIRALLES

# Índice

<b>Almacenamiento Web</b>	<b>3</b>
Introducción	3
Almacenamiento en sesión	4
Ejemplo	4
Almacenamiento en local	7
Ejemplo	8
Otras opciones de almacenamiento	10
Almacenamiento en caché	10
<b>Ejercicios</b>	<b>13</b>
Ejercicio 1: Crear un carrito de la compra que vaya almacenando el listado de productos en local	13
Lo necesario para comenzar	14
Pasos a seguir	14
Solución del Ejercicio	15

# Almacenamiento Web

## Introducción

El almacenamiento Web, es una de las opciones más novedosas que HTML5 presenta. Hasta el momento, era imposible almacenar información para que estuviera disponible en otro momento, si no lo hacíamos mediante un lenguaje de servidor y una base de datos. Pues bien, mediante HTML5 y un lenguaje de script, en este caso JavaScript, vamos a tener la posibilidad de almacenar información de manera local, sin necesidad de utilizar ningún lenguaje de servidor. Esta es una de las necesidades que más echaban de menos los desarrolladores de contenido Web, ya que facilita mucho las cosas a la hora de mantener almacenados ciertos datos, evita muchas conexiones con el servidor, y además, abre las puertas a crear aplicaciones Web que puedan funcionar *offline*.

Hay que dejar bien claro, que estamos hablando de almacenamiento en local, y que lo haremos a través de lenguajes de lado de cliente como son HTML y JavaScript. Por lo tanto, los datos almacenados estarán disponibles para el propio usuario únicamente, si queremos que lo que se almacene esté disponible para todos los usuarios que se vayan conectando, necesitamos por obligación, un lenguaje de servidor y una base de datos.

Hasta el momento, cuando actualizábamos nuestra página, o volvíamos a cargarla, todas las variables eran reiniciadas y se perdía cualquier tipo de información que con anterioridad se había obtenido. A partir de ahora, podremos almacenar dicha información y tenerla disponible en cualquier otro momento, o mientras dure una sesión. El almacenamiento por lo tanto, se podrá hacer de dos maneras diferentes, en la propia sesión activa del navegador, o de manera local en el propio navegador. En ambos casos, el almacenamiento se hará como pares clave y valor, es decir, se asignará un nombre o clave a cada dato almacenado.

Además de estas dos posibilidades para almacenar datos en local, tenemos disponibles otras opciones que veremos más adelante.



### ¡IMPORTANTE!

Como hemos dicho, estamos hablando de almacenamiento en local, por lo tanto los datos almacenados estarán únicamente disponibles para el propio usuario. Si necesitáramos almacenar información para que estuviera disponible para todos los usuarios, obligatoriamente necesitamos un lenguaje de servidor y una base de datos.



**¡IMPORTANTE!**

Podemos consultar la recomendación del W3C entorno a Web Storage desde [aquí](#).



**¡IMPORTANTE!**

El almacenamiento Web, está en funcionamiento en los siguientes navegadores:



Para su correcto funcionamiento, es recomendable ejecutarlo siempre desde un servidor y no en local.

## Almacenamiento en sesión

Vamos a comenzar con el almacenamiento en la propia sesión del navegador. Los datos almacenados de esta manera, estarán disponibles mientras dure la sesión, pero si durante 15 minutos el navegador no detecta ningún movimiento en la página, la sesión terminará y estos datos se borrarán.

El objeto que vamos a utilizar para este tipo de almacenamiento es *sessionStorage*, que dispone de una serie de atributos y métodos que vamos a ver a continuación:

- **length:** Atributo que indica el numero de pares clave/valor que tenemos almacenados en el objeto *sessionStorage*.
- **key(n):** Función que devuelve el nombre de la clave que se encuentra almacenada en la posición *n*.
- **getItem(clave):** Función que devuelve el valor de la clave indicada.
- **setItem(clave, valor):** Función que almacena en el objeto el par clave/valor indicado.
- **removeItem(clave):** Función que elimina el par clave/valor de la clave indicada.
- **clear():** Función que limpia el objeto *sessionStorage* y lo deja vacío.

Utilizando este objeto junto con sus métodos y atributos, podemos ir almacenando datos y utilizándolos mientras dure la sesión. A continuación vamos a ver un ejemplo donde vamos a observar claramente su funcionamiento:

## Ejemplo

En el siguiente ejemplo vamos a ver el funcionamiento de todos los métodos y atributos del objeto *sessionStorage*. Vamos a poder ir almacenando datos en la sesión del navegador, y en cualquier momento, mientras dure la sesión, podremos consultar o borrar uno de los datos almacenados, o el listado completo. Lo que queremos conseguir es lo siguiente:

Motos	Guardar		
0	Ver elemento	Ver listado	
0	Borrar elemento	Borrar listado	

Elemento0 = Coches  
 Elemento1 = Furgonetas  
 Elemento2 = Motos

Vemos en primer lugar que podemos escribir un dato, y haciendo clic sobre el botón de guardar, lo almacenaremos en la sesión del navegador. Tenemos también la opción de mostrar solo el dato que está en la posición indicada en el campo de entrada numérico, o de mostrar todo el listado almacenado. Y de la misma manera, tenemos la opción de borrar solo el dato que está en la posición indicada en el campo de entrada numérico, o de borrar todo el listado almacenado.

Vamos en primer lugar con el código HTML de nuestra página, que será el siguiente:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Ejemplo sessionStorage</title>
5
6  <style>
7    #listado, #botones{
8      border:2px solid black;
9      width:414px;
10     margin:5px auto;
11   }
12   #listado{
13     min-height:200px;
14   }
15 </style>
16 </head>
17
18 <body>
19   <div id="botones">
20     <input type="text" id="elemento">
21     <input type="button" value="Guardar" onclick="almacenar()"><br>
22
23     <input type="number" value="0" id="verEl">
24     <input type="button" value="Ver elemento" onclick="verElemento()">
25     <input type="button" value="Ver listado" onclick="ver()"><br>
26
27     <input type="number" value="0" id="borrEl">
28     <input type="button" value="Borrar elemento" onclick="borrarElemento()">
29     <input type="button" value="Borrar listado" onclick="borrar()">
30   </div>
31   <div id="listado"></div>
32 </body>
33 </html>

```

Hemos creado los botones y campos de entrada necesarios para conseguir lo indicado anteriormente, y le hemos aplicado el formato deseado mediante la etiqueta *style*. Cada botón creado, tiene una función asignada, que ejecutará en el momento que hagamos clic sobre él. A continuación, vamos a ir viendo cada una de esas funciones.

Comenzamos por la función *almacenar()*, que será la encargada de recoger el contenido del elemento de entrada con identificador "elemento" y almacenarlo en la sesión del navegador. El código sería el siguiente:

```

19  var cantidad = sessionStorage.length;
20
21  function almacenar() {
22      var contenido = document.getElementById("elemento").value;
23      sessionStorage.setItem("Elemento"+cantidad, contenido);
24      cantidad++;
25  }

```

La variable "cantidad", llevará el control de los elementos que hemos almacenado, cuando cargue la página, cogerá el valor de la cantidad de elementos almacenados ya, y se irá aumentando cada vez que almacenemos un nuevo valor. Este valor lo necesitamos, ya que la clave para cada uno de los valores almacenados será la palabra "Elemento" seguida de un número, y por tanto, en función de los valores que ya hayamos almacenado, le asignaremos el siguiente número al nuevo valor que almacenemos. Una vez disponemos de ese valor, recogemos el contenido del campo de entrada con identificador "elemento", y lo almacenamos de la manera que hemos explicado utilizando el método *setItem()*.

La siguiente función será *verElemento()*, que será la encargada de mostrar en el *DIV* con identificador "listado", el elemento que se encuentre almacenado en la posición que indiquemos en el campo de entrada de tipo numérico con identificador "verEl". El código sería el siguiente:

```

25  function verElemento() {
26      var posicion = document.getElementById("verEl").value;
27
28      var clave = sessionStorage.key(posicion);
29      var dato = sessionStorage.getItem(clave);
30      document.getElementById("listado").innerHTML = clave+" = "+dato;
31  }

```

Recogemos el valor del campo de entrada numérico con identificador "verEl", y con dicho valor mediante el método *key()*, consultamos cuál es la clave del elemento almacenado en dicha posición. Una vez que disponemos de la clave del elemento, solo nos queda extraerlo del objeto mediante el método *getItem()*, y mostrar tanto la clave como el valor en el *DIV* con identificador "listado".

La función *ver()*, será la encargada de mostrar en el *DIV* con identificador "listado", el listado completo de elementos almacenados en el objeto *sessionStorage*. El código sería el siguiente:

```

35  function ver() {
36      var cont = sessionStorage.length;
37      document.getElementById("listado").innerHTML = "";
38      for(i=0;i<cont;i++)
39      {
40          var clave = sessionStorage.key(i);
41          var dato = sessionStorage.getItem(clave);
42          document.getElementById("listado").innerHTML += clave+" = "+dato+"<br>";
43      }
44  }

```

En este caso, debemos mostrar el listado completo de elementos almacenados en el objeto *sessionStorage*. Para ello, lo primero que hacemos es conocer cuántos elementos tenemos almacenados mediante el atributo *length*. Después, vaciamos el *DIV* con identificador "listado", ya que será ahí donde mostremos el listado completo. Una vez hecho esto, crearemos un bucle *for*, que irá extrayendo y mostrando uno por uno todos los elementos de la lista. Primero extraemos la clave de cada posición con el método *key()*, después extraemos el valor mediante el método *getItem()* y finalmente añadimos tanto clave como valor al *DIV* donde queremos mostrarlo.

La siguiente función será *borrarElemento()*, que será la encargada de borrar el elemento que se encuentre almacenado en la posición que indiquemos en el campo de entrada de tipo numérico con identificador "borrEl". El código sería el siguiente:

```
43 function borrarElemento() {
44     var posicion = document.getElementById("borrEl").value;
45
46     var clave = sessionStorage.key(posicion);
47     var dato = sessionStorage.removeItem(clave);
48 }
```

Recogemos el valor del campo de entrada numérico con identificador "borrEl", y con dicho valor mediante el método *key()*, consultamos cuál es la clave del elemento almacenado en dicha posición. Una vez que disponemos de la clave del elemento, solo nos queda eliminarlo del objeto mediante el método *removeItem()*.

La función *borrar()*, será la encargada de borrar el listado completo de elementos almacenados en el objeto *sessionStorage*. El código sería el siguiente:

```
50 function borrar() {
51     sessionStorage.clear();
52 }
```

Para borrar el listado completo, únicamente utilizaremos el método *clear()*, que se encarga de vaciar el objeto *sessionStorage*.



Ejemplo de almacenamiento en la sesión del navegador

## Almacenamiento en local

En este caso, el almacenamiento se hará de manera local en el propio navegador. Los datos almacenados de esta manera, estarán disponibles en ese navegador hasta que se borren.

El objeto que vamos a utilizar para este tipo de almacenamiento es *localStorage*, que dispone de una serie de atributos y métodos que vamos a ver a continuación:

- **length:** Atributo que indica el número de clave/valor que tenemos almacenados en el objeto *localStorage*.
- **key(n):** Función que devuelve el nombre de la clave que se encuentra almacenada en la posición *n*.
- **getItem(clave):** Función que devuelve el valor de la clave indicada.
- **setItem(clave, valor):** Función que almacena en el objeto el par clave/valor indicado.

- **removeItem(clave):** Función que elimina el par clave/valor de la clave indicada.
- **clear():** Función que limpia el objeto `localStorage` y lo deja vacío.

Utilizando este objeto junto con sus métodos y atributos, podemos ir almacenando datos en el navegador y utilizándolos hasta que sean borrados. A continuación vamos a ver un ejemplo donde vamos a ver más claramente su funcionamiento:

### Ejemplo

El ejemplo para el almacenamiento en local del propio navegador, será exactamente igual que el que hemos visto en el apartado de almacenamiento en sesión, pero utilizando el objeto `localStorage` en lugar de `sessionStorage`. El código completo quedaría de la siguiente manera:



```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Ejemplo localStorage</title>
5
6      <style>
7          #listado, #botones{
8              border:2px solid black;
9              width:414px;
10             margin:5px auto;
11         }
12         #listado{
13             min-height:200px;
14         }
15     </style>
16 </head>
17 <script type="text/javascript">
18
19     var cantidad = localStorage.length;
20
21     function almacenar() {
22         var contenido = document.getElementById("elemento").value;
23         localStorage.setItem("Elemento"+cantidad, contenido);
24         cantidad++;
25     }
26
27     function verElemento() {
28         var posicion = document.getElementById("verEl").value;
29
30         var clave = localStorage.key(posicion);
31         var dato = localStorage.getItem(clave);
32         document.getElementById("listado").innerHTML = clave+" = "+dato;
33     }
34
35     function ver() {
36         var cont = localStorage.length;
37         document.getElementById("listado").innerHTML = "";
38         for(i=0;i<cont;i++)
39         {
40             var clave = localStorage.key(i);
41             var dato = localStorage.getItem(clave);
42             document.getElementById("listado").innerHTML += clave+" = "+dato+"<br>";
43         }
44     }
45
46     function borrarElemento() {
47         var posicion = document.getElementById("borrEl").value;
48
49         var clave = localStorage.key(posicion);
50         var dato = localStorage.removeItem(clave);
51     }
52
53     function borrar() {
54         localStorage.clear();
55     }
56
57 </script>
58 <body>
59     <div id="botones">
60         <input type="text" id="elemento">
61         <input type="button" value="Guardar" onclick="almacenar()"><br>
62
63         <input type="number" value="0" id="verEl">
64         <input type="button" value="Ver elemento" onclick="verElemento()">
65         <input type="button" value="Ver listado" onclick="ver()"><br>
66
67         <input type="number" value="0" id="borrEl">
68         <input type="button" value="Borrar elemento" onclick="borrarElemento()">
69         <input type="button" value="Borrar listado" onclick="borrar()">
70     </div>
71     <div id="listado"></div>
72 </body>
73 </html>

```

Vemos como todo es absolutamente igual, solo que en este caso utilizamos el objeto *localStorage*. Por lo tanto, en este caso, los datos estarán disponibles en el navegador hasta que sean borrados y no hasta que finalice la sesión.



Ejemplo de almacenamiento en local en el propio navegador

## Otras opciones de almacenamiento

La necesidad de almacenar datos en local es más que evidente, por eso, HTML5 ofrece otras posibilidades para almacenar datos en el cliente, aparte de las que ya hemos visto. Entre ellas, nos encontramos con estas otras tres opciones:

- **Almacenamiento Web en BD SQL:** Proporciona toda la potencia de una base de datos relacional SQL estructurada, el almacenamiento y extracción de los datos se realiza haciendo consultas SQL, idénticas a las que haríamos en servidor. En este caso, el desarrollo de la API en la que W3C trabajaba, se detuvo ya que no llegaban a ningún punto concreto en la estandarización. Podemos consultar la recomendación de la W3C desde este enlace: [Web SQL Database](#).
- **Bases de datos indexadas:** Está a medio camino entre el almacenamiento Web y el almacenamiento Web en BD SQL. Al igual que el almacenamiento Web, se trata de asignaciones de valores y claves, pero en este caso se admiten índices similares a los de las bases de datos relacionales. Podemos consultar la recomendación de la W3C desde este enlace: [Indexed Database API](#).
- **Almacenamiento en ficheros:** Esta opción, nos da la posibilidad de guardar contenido binario (como texto plano) creando carpetas y jerarquía de las mismas permitiéndonos de esta forma guardar grandes estructuras de datos. Podemos consultar la recomendación de la W3C desde este enlace: [File API](#).

Como vemos, en HTML5 se ofrecen varias posibilidades para almacenar información en el cliente, lo que abre las puertas a crear aplicaciones Web que puedan funcionar en el lado del usuario, sin cargar al servidor. En este proceso, es interesante también, que estas aplicaciones puedan funcionar de manera *offline*, es decir, sin necesitar una conexión a internet.

## Almacenamiento en caché

Ya hemos visto como ir almacenando datos en local, por lo tanto, ahora solo nos queda que la aplicación funcione sin hacer uso del servidor. Para ello, podemos indicarle a la aplicación Web, qué ficheros debe de almacenar en caché, para que todo funcione de manera correcta cuando no dispongamos de una conexión a internet. Podemos pensar que para eso está la memoria caché automática del navegador, pero ésta no nos garantiza el almacenamiento de todos los ficheros necesarios.

De esta manera, aparte de ofrecer la posibilidad de utilizar la aplicación *offline*, ganamos en velocidad y quitamos carga de servidor, que irá descargándose los elementos solo cuando vayan actualizándose. Para conseguir esto, debemos de crear un fichero con extensión recomendada **.appcache** y referenciarlo desde nuestro fichero HTML. Vamos a verlo claramente a continuación.

En primer lugar, debemos de referenciar el fichero donde tendremos el listado con todos los elementos a almacenar en caché, esto se hará mediante el atributo *manifest* de la etiqueta *<html>* de la siguiente manera:

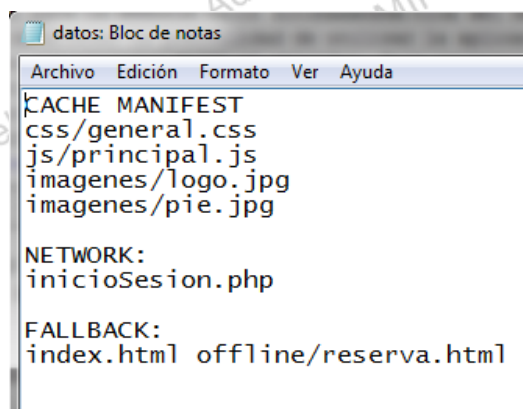
```
1 <!DOCTYPE html>
2 <html manifest="datos.appcache">
```

Vemos como hemos referenciado el fichero *datos.appcache* para el documento. La estructura de dicho fichero la veremos a continuación.

Este archivo, deberá ser tratado con el tipo de dato *MIME* correcto, que es, *"text/cache-manifest"* y se deberá configurar de manera correcta en el servidor que lo coloquemos. En cuanto a la estructura de este, puede tener tres apartados:

- **CACHE MANIFEST:** En este apartado, enumeraremos la lista de ficheros que deben de descargarse a la caché.
- **NETWORK:** En este apartado, enumeraremos la lista de ficheros que requieren una conexión con el servidor y que nunca deben de descargarse a la caché.
- **FALLBACK:** En este apartado, enumeraremos las páginas de reserva por si alguna página es inaccesible.

Por lo tanto, el fichero *datos.appcache* podría ser el siguiente:



```
datos: Bloc de notas
Archivo Edición Formato Ver Ayuda

CACHE MANIFEST
css/general.css
js/principal.js
imagenes/logo.jpg
imagenes/pie.jpg

NETWORK:
inicioSesion.php

FALLBACK:
index.html offline/reserva.html
```

Hemos indicado que cuando se cargue por primera vez la página, debe de descargar el fichero *general.css* que se encuentra en la carpeta *css*, el fichero *principal.js* que se encuentra en la carpeta *js*, y las imágenes *logo.jpg* y *pie.jpg* que se encuentran en la carpeta *imagenes* a la caché, para tenerlas disponibles también *offline*. Por otro lado, hemos indicado que el fichero *inicioSesion.php* requiere de una conexión al servidor y que nunca debe de descargarse a la caché, y hemos indicado también, que en caso de que la página principal no este accesible, cargue la página de reserva que se encuentra en la carpeta *offline*. Hay que destacar, que la página principal no se añade al apartado *CACHE MANIFEST*, ya que la descarga de la misma a la caché se hace por defecto.

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

Escuela Riojana de Administración Pública  
NURIA ARAGON MIRALLES

de Administración Pública  
NURIA ARAGON MIRALLES



## Ejercicios

### Ejercicio 1: Crear un carrito de la compra que vaya almacenando el listado de productos en local



Vamos a crear un sistema que nos permita simular un carrito de la compra. El objetivo será tener un listado de productos publicados, y que podamos ir añadiendo diferentes cantidades de cada producto a nuestro carrito. En caso de que dicho producto ya este en la lista, se sumará la cantidad seleccionada a dicho producto y no se duplicará. Según vayamos añadiendo cosas al carrito, este se irá actualizando, así como el importe total de nuestra compra. Desde el listado, podremos también eliminar cualquier producto que ya no queramos, y en ese momento el carrito y el precio total también se actualizarán. Nuestro listado, se irá almacenando de manera local, y estará activo el tiempo que dure nuestra sesión. Lo que queremos conseguir es lo que vemos en la siguiente imagen:

## TIENDA ONLINE

IMGEN	DESCRIPCION	PRECIO	CANTIDAD	COMPRAR
	Ordenador DELL	987€	<input type="text" value="1"/>	<input type="button" value="Comprar"/>
	Ordenador SONY	787€	<input type="text" value="3"/>	<input type="button" value="Comprar"/>
	Ordenador APPLE	1073€	<input type="text" value="1"/>	<input type="button" value="Comprar"/>
	Ordenador PBELL	585€	<input type="text" value="1"/>	<input type="button" value="Comprar"/>
	Ordenador ACER	889€	<input type="text" value="1"/>	<input type="button" value="Comprar"/>

**Este es el estado de tu carrito**

PRO	PRECIO	UNDS	TOTAL	
Ordenador SONY	787	3	2361	
Ordenador APPLE	1073	1	1073	
<b>PRECIO TOTAL DE LA COMPRA</b>				
3434€				

## Lo necesario para comenzar



**imagenes.zip**

---

Descarga las imágenes que utilizamos para llevar a la práctica el ejercicio

## Pasos a seguir

1. En primer lugar, estructuraremos la página para conseguir el formato que vemos en la imagen. El listado de productos esta dentro de una tabla, y los campos de entrada para seleccionar la cantidad son de tipo *number*. El contenido del carrito, se creará dinámicamente.
2. Una vez estructurada la página, tendremos que crear las funciones de JavaScript que llevarán a cabo todo el proceso que hemos indicado anteriormente.
3. Los productos que vayamos seleccionando se irán almacenando en el objeto de almacenamiento en la sesión del navegador, y los que vayamos eliminando se quitarán de la misma.
4. El carrito, lo generará una función, dependiendo del contenido del objeto de almacenamiento que hemos indicado anteriormente.

## Solución del Ejercicio



Crear un carrito de la compra que vaya almacenando el listado de productos en local (Parte 1)

Crear un carrito de la compra que vaya almacenando el listado de productos en local (Parte 2)