sessionStorage. Este es un mecanismo de almacenamiento que conservará los datos disponibles solo durante la duración de la sesión de una página. De hecho, a diferencia de sesiones reales, la información almacenada a través de este mecanismo es solo accesible desde una única ventana o pestaña y es preservada hasta que la ventana es cerrada. La especificación aún nombra "sesiones" debido a que la información es preservada incluso cuando la ventana es actualizada o una nueva página desde el mismo sitio web es cargada.

localStorage. Este mecanismo trabaja de forma similar a un sistema de almacenamiento para aplicaciones de escritorio. Los datos son grabados de forma permanente y se encuentran siempre disponibles para la aplicación que los creó.

Ambos mecanismos trabajan a través de la misma interfaz, compartiendo los mismos métodos y propiedades. Y ambos son dependientes del origen, lo que quiere decir que la información está disponible solo a través del sitio web o la aplicación que los creó. Cada sitio web tendrá designado su propio espacio de almacenamiento que durará hasta que la ventana es cerrada o será permanente, de acuerdo al mecanismo utilizado.

La API claramente diferencia datos temporarios de permanentes, facilitando la construcción de pequeñas aplicaciones que necesitan preservar solo unas cadenas de texto como referencia temporaria (por ejemplo, carros de compra) o aplicaciones más grandes y complejas que necesitan almacenar documentos completos por todo el tiempo que sea necesario.

IMPORTANTE:
Muchos navegadores solo trabajan de forma adecuada cuando la fuente es un servidor real. Para probar los siguientes códigos, le recomendamos que primero suba los archivos a su servidor o ejecute con liveserver.

Implementación de un sistema de almacenamiento de datos

Debido a que ambos sistemas, **sessionStorage** y **localStorage**, trabajan con la misma interfaz, vamos a necesitar solo un documento HTML y un simple formulario para probar los códigos y experimentar:

```
<!DOCTYPE html>
<html lang="es">

<head>
  <title>Web Storage API</title>
  <link rel="stylesheet" href="storage.css">
  <script src="storage.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p>Clave:<br><input type="text" name="clave" id="clave"></p>
      <p>Valor:<br><textarea name="text" id="texto"></textarea></p>
      <p><input type="button" name="grabar" id="grabar"
                                              value="Grabar"></p>
    </form>
  </section>
  <section id="cajadatos">
    No hay información disponible
  </section>
</body>
</html>
```

También crearemos un grupo de reglas de estilo simples para dar forma a la página y diferenciar el área del formulario de la caja donde los datos serán mostrados y listados:

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto{
  width: 200px;
}
#cajadatos > div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

Creando datos

Ambos, **sessionStorage** y **localStorage**, almacenan datos como ítems. Los ítems están formados por un par clave/valor, y cada valor será convertido en una cadena de texto antes de ser almacenado. Piense en ítems como si fueran variables, con un nombre y un valor, que pueden ser creadas, modificadas o eliminadas.
Existen dos nuevos métodos específicos de esta API incluidos para crear y leer un valor en el espacio de almacenamiento:

setItem(clave, valor) Este es el método que tenemos que llamar para crear un ítem. El ítem será creado con una clave y un valor de acuerdo a los atributos especificados. Si ya existe un ítem con la misma clave, será actualizado al nuevo valor, por lo que este método puede utilizarse también para modificar datos previos.
getItem(clave) Para obtener el valor de un ítem, debemos llamar a este método especificando la clave del ítem que queremos leer. La clave en este caso es la misma que declaramos cuando creamos al ítem con **setItem()**.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;
  sessionStorage.setItem(clave,valor);

  mostrar(clave);
}
function mostrar(clave){
  var cajadatos=document.getElementById('cajadatos');
 var valor=sessionStorage.getItem(clave);
  cajadatos.innerHTML='<div>'+clave+' - '+valor+'</div>';
}
window.addEventListener('load', iniciar, false);
```

El proceso es extremadamente simple. Los métodos son parte de **sessionStorage** y son llamados con la sintaxis **sessionStorage.setItem()**. La función **nuevoitem()** es ejecutada cada vez que el usuario hace clic en el botón del formulario. Esta función crea un ítem con la información insertada en los campos del formulario y luego llama a la función **mostrar()**. Esta última función lee el ítem de acuerdo a la clave recibida usando el método **getItem()** y muestra su valor en la pantalla.

Además de estos métodos, la API también ofrece una sintaxis abreviada para crear y leer ítems desde el espacio de almacenamiento. Podemos usar la clave del ítem como una propiedad y acceder a su valor de esta manera.
Este método usa en realidad dos tipos de sintaxis diferentes de acuerdo al tipo de información que estamos usando para crear el ítem. Podemos encerrar una variable representando la clave entre corchetes (por ejemplo, **sessionStorage[clave]=valor**) o podemos usar directamente el nombre de la propiedad (por ejemplo, **sessionStorage.miitem=valor)**.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;
 sessionStorage[clave]=valor;

  mostrar(clave);
}
function mostrar(clave){
  var cajadatos=document.getElementById('cajadatos');
 var valor=sessionStorage[clave];
  cajadatos.innerHTML='<div>'+clave+' - '+valor+'</div>';
}
window.addEventListener('load', iniciar, false);
```

Leyendo datos

El anterior ejemplo solo lee el último ítem grabado. Vamos a mejorar este código aprovechando más métodos y propiedades provistos por la API con el propósito de manipular ítems:

Length Esta propiedad retorna el número de ítems guardados por esta aplicación en el espacio de almacenamiento. Trabaja exactamente como la propiedad **length** usada normalmente en Javascript para procesar arrays, y es útil para lecturas secuenciales.

key(índice) Los ítems son almacenados secuencialmente, enumerados con un índice automático que comienzo por 0. Con este método podemos leer un ítem específico o crear un bucle para obtener toda la información almacenada.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
  mostrar();
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;

  sessionStorage.setItem(clave,valor);
  mostrar();
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
```

```
}
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML='';
 for(var f=0;f<sessionStorage.length;f++){
  var clave=sessionStorage.key(f);
  var valor=sessionStorage.getItem(clave);
  cajadatos.innerHTML+='<div>'+clave+' - '+valor+'</div>';
 }
}
window.addEventListener('load', iniciar, false);
```

El propósito del código es mostrar un listado completo de los ítems en la caja derecha de la pantalla. La función **mostrar()** fue mejorada usando la propiedad **length** y el método **key()**. Creamos un bucle **for** que va desde 0 al número de ítems que existen en el espacio de almacenamiento. Dentro del bucle, el método **key()** retornará la clave que nosotros definimos para cada ítem. Por ejemplo, si el ítem en la posición 0 del espacio de almacenamiento fue creado con la clave "miitem", el código **sessionStorage.key(0)** retornará el valor "miitem". Llamando a este método desde un bucle podemos listar todos los ítems en la pantalla con sus correspondientes claves y valores.
La función **mostrar()** es llamada desde la función **iniciar()** tan pronto como la aplicación es ejecutada. De este modo podremos ver desde el comienzo los ítems que fueron grabados previamente en el espacio de almacenamiento.

Eliminando datos

Los ítems pueden ser creados, leídos y, por supuesto, eliminados. Es hora de ver cómo eliminar un ítem.
La API ofrece dos métodos para este propósito:
removeItem(clave) Este método eliminará un ítem individual. La clave para identificar el ítem es la misma declarada cuando el ítem fue creado con el método **setItem()**.
clear() Este método vaciará el espacio de almacenamiento. Todos los ítems serán eliminados.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
  mostrar();
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;

  sessionStorage.setItem(clave,valor);
  mostrar();
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
}
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML='<div><button
     onclick="eliminarTodo()">Eliminar Todo</button></div>';
  for(var f=0;f<sessionStorage.length;f++){
    var clave=sessionStorage.key(f);
    var valor=sessionStorage.getItem(clave);
    cajadatos.innerHTML+='<div>'+clave+' - '+valor+'<br><button
      onclick="eliminar(\''+clave+'\')">Eliminar</button></div>';
  }
}
function eliminar(clave){
  if(confirm('Está Seguro?')){
  sessionStorage.removeItem(clave);
    mostrar();
```

```
  }
}
function eliminarTodo(){
  if(confirm('Está Seguro?')){
  sessionStorage.clear();
    mostrar();
  }
}
window.addEventListener('load', iniciar, false);
```

Las funciones **iniciar()** y **nuevoitem()** son las mismas de códigos previos. Solo la función **mostrar()** cambia para incorporar el manejador de eventos **onclick** y llamar a las funciones que eliminarán un ítem individual o vaciarán el espacio de almacenamiento. La lista de ítems presentada en pantalla es construida de la misma manera que antes, pero esta vez un botón "Eliminar" es agregado junto a cada ítem para poder eliminarlo. Un botón para eliminar todos los ítems juntos también fue agregado en la parte superior.
Las funciones **eliminar()** y **eliminarTodo()** se encargan de eliminar el ítem seleccionado o limpiar el espacio de almacenamiento, respectivamente. Cada función llama a la función **mostrar()** al final para actualizar la lista de ítems en pantalla.

El sistema **sessionStorage** preserva los datos creados en una ventana solo hasta que esa ventana es cerrada. Es útil para controlar carros de compra o cualquier otra aplicación que requiere acceso a datos por períodos cortos de tiempo.

localStorage

Disponer de un sistema confiable para almacenar datos durante la sesión de una ventana puede ser extremadamente útil en algunas circunstancias, pero cuando intentamos simular poderosas aplicaciones de escritorio en la web, un sistema de almacenamiento temporario no es suficiente.
Para cubrir este aspecto, Storage API ofrece un segundo sistema que reservará un espacio de almacenamiento para cada aplicación (cada origen) y mantendrá la información disponible permanentemente. Con **localStorage**, finalmente podemos grabar largas cantidades de datos y dejar que el usuario decida si la información es útil y debe ser conservada o no.
El sistema usa la misma interfaz que **sessionStorage**, debido a esto cada método y propiedad estudiada hasta el momento está también disponible para **localStorage**. Solo la substitución del prefijo **session** por **local** es requerida para preparar los códigos.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
  mostrar();
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;

 localStorage.setItem(clave,valor);
  mostrar();
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
}
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML='';
  for(var f=0;f<localStorage.length;f++){
    var clave=localStorage.key(f);
    var valor=localStorage.getItem(clave);
    cajadatos.innerHTML+='<div>'+clave+' - '+valor+'</div>';
  }
```

```
}
window.addEventListener('load', iniciar, false);
```

Simplemente reemplazamos **sessionStorage** por **localStorage** en el código de uno de los ejemplos anteriores. Ahora, cada ítem creado será preservado a través de diferentes ventanas e incluso después de que todas las ventanas del navegador son cerradas.

Evento storage

Debido a que **localStorage** hace que la información esté disponible en cada ventana donde la aplicación fue cargada, surgen al menos dos problemas: debemos resolver cómo estas ventanas se comunicarán entre sí y cómo haremos para mantener la información actualizada en cada una de ellas. En respuesta a ambos problemas, la especificación incluye el evento **storage**.

Storage Este evento será disparado por la ventana cada vez que un cambio ocurra en el espacio de almacenamiento. Puede ser usado para informar a cada ventana abierta con la misma aplicación que los datos han cambiado en el espacio de almacenamiento y que se debe hacer algo al respecto.

```
function iniciar(){
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', nuevoitem, false);
  window.addEventListener("storage", mostrar, false);

  mostrar();
}
function nuevoitem(){
  var clave=document.getElementById('clave').value;
  var valor=document.getElementById('texto').value;

  localStorage.setItem(clave,valor);
  mostrar();
  document.getElementById('clave').value='';
  document.getElementById('texto').value='';
}
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML='';
  for(var f=0;f<localStorage.length;f++){
    var clave=localStorage.key(f);
    var valor=localStorage.getItem(clave);
    cajadatos.innerHTML+='<div>'+clave+' - '+valor+'</div>';
  }
}
window.addEventListener('load', iniciar, false);
```

Solo necesitamos comenzar a escuchar al evento **storage** en la función **iniciar()** para ejecutar la función **mostrar()** en cada ventana siempre que un ítem es creado, modificado o eliminado. Ahora, si algo cambia en una ventana, el cambio será mostrado automáticamente en el resto de las ventanas que están ejecutando la misma aplicación.

Espacio de almacenamiento

La información almacenada por **localStorage** será permanente a menos que el usuario decida que ya no la necesita. Esto significa que el espacio físico en el disco duro ocupado por esta información probablemente crecerá cada vez que la aplicación sea usada. Hasta este momento, la especificación de HTML5 recomienda a los fabricantes de navegadores que reserven un mínimo de 5 megabytes para cada origen (cada sitio web o aplicación). Esta es

solo una recomendación que probablemente cambiará dramáticamente en los próximos años. Algunos navegadores están consultando al usuario si expandir o no el espacio disponible cuando la aplicación lo necesita, pero usted debería ser consciente de esta limitación y tenerla en cuenta a la hora de desarrollar sus aplicaciones.

Referencia rápida

Con la ayuda de la API Web Storage, ahora las aplicaciones web pueden ofrecer un espacio de almacenamiento. Usando un par clave/valor, la información es almacenada en el ordenador del usuario para un rápido acceso o para trabajar desconectado de la red.

Tipo de almacenamiento

Dos mecanismos diferentes son ofrecidos para almacenar datos:

sessionStorage Este mecanismo mantiene la información almacenada solo disponible para una simple ventana y solo hasta que la ventana es cerrada.

localStorage Este mecanismo almacena datos de forma permanente. Estos datos son compartidos por todas las ventanas que están ejecutando la misma aplicación y estarán disponibles a menos que el usuario decida que ya no los necesita.

Métodos

La API incluye una interface común para cada mecanismo que cuenta con nuevos métodos, propiedades y eventos:

setItem(clave, valor) Este método crea un nuevo ítem que es almacenado en el espacio de almacenamiento reservado para la aplicación. El ítem está compuesto por un par clave/valor creado a partir de los atributos **clave** y **valor**.

getItem(clave) Este método lee el contenido de un ítem identificado por la clave especificada por el atributo **clave**. El valor de esta clave debe ser el mismo usado cuando el ítem fue creado con el método **setItem()**.

key(índice) Este método retorna la clave del ítem encontrado en la posición especificada por el atributo **índice** dentro del espacio de almacenamiento.

removeItem(clave) Este método elimina un ítem con la clave especificada por el atributo **clave**. El valor de esta clave debe ser el mismo usado cuando el ítem fue creado por el método **setItem()**.

clear() Este método elimina todos los ítems en el espacio de almacenamiento reservado para la aplicación.

Propiedades

length Esta propiedad retorna el número de ítems disponibles en el espacio de almacenamiento reservado para la aplicación.

Eventos

Storage Este evento es disparado cada vez que un cambio se produce en el espacio de almacenamiento reservado para la aplicación.