UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo de aplicaciones para moviles

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Módulo 2:

Desarrollo de aplicaciones

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Unidad 6:

Almacenamiento local: Local y Session Storage

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación de la Unidad:

En esta unidad estaremos viendo como almacenar datos con webstorage y sesión storage, que es esto, imaginemos que debemos almacenar algunos datos y no contamos con una base de datos para poder hacerlo.
Tenemos las opciones de guardarlo en una variable de sesión que deberíamos contar con la ayuda del servidor, o de un webstorage, sería básicamente almacenarlo en el navegador

Objetivos:

Unidad 6: Saber cómo almacenar información dentro de la aplicación, para este ejemplo decidí utilizar Jquery ya que estaremos viendo más adelante como crear aplicaciones con jquery Mobile , una alternativa al ionic y phonegap.

BLOQUES TEMÁTICOS:

Contenido

Qué es localStorage

Dentro de las múltiples novedades que existen en HTML5, una de ellas es el **localStorage**. Como su propio nombre indica, se trata de un espacio de almacenamiento local. A muchos les vendrá a la mente las cookies…

Bien, cierto es que con las cookies ya podemos almacenar información en el equipo que accede a la página web… que es exactamente de lo que se trata el **localStorage**, pero con una serie de salvedades muy importantes.

LOCALSTORAGE VS COOKIES

La mejor forma de entender por qué es necesario el **localStorage** es indicando los tres grandes problemas de las cookies:

1. ***Espacio limitado:*** Una cookie sólo puede ocupar 4kb de espacio. Es por eso que las cookies suelen utilizarse sólo para almacenar un hash o un identificador que será utilizado por el servidor para identificar la visita.
2. Cada vez que se realiza una petición al servidor, toda la información que está almacenada en las ***cookies*** es enviada y también es recibida nuevamente con la respuesta del servidor. O sea, en los intercambios de información entre el navegador web y el servidor siempre van pegadas las cookies.
3. Las cookies tienen una ***caducidad***.

Y aquí viene **localStorage** a solucionarlos la vida

1. Espacio menos limitado: ***localStorage puede ocupar entre 5 y 10MB*** dependiendo del navegador web. Con 5 o 10 megas ya podemos tener algo más de información
2. La información almacenada con localStorage ***no es enviada al servidor en cada petición***.
3. ***No existe una caducidad para localStorage***, la información quedará almacenada hasta que se elimine expresamente. Aunque se cierre el navegador.

Lo sé, una cookie tiene caducidad, pero le puedes poner que caduque dentro de 5 años… vale, sí… pero caduca, con localStorage nos olvidamos de tener que guardar la cookie aumentando el tiempo de caducidad.

Sin embargo, que la información persista en el tiempo, no siempre es una buena idea. A veces lo que interesa es que la información se elimina una vez se cierre el navegador. Para estos casos, en vez de utilizar localStorage, se debe usar ***sessionStorage***.

El ***sessionStorage*** es exactamente igual que ***localStorage***, pero con la salvedad de que una vez cerrado el navegador se pierde la información, todo lo demás es lo mismo.

LA API DEL LOCALSTORAGE

HTML5 incluye una **API de JavaScript** para interactuar con **localStorage**, teniendo 4 métodos disponibles para usar con nuestro **localStorage**.

ALMACENAR UN VALOR LOCALSTORAGE

Para guardar la información utilizamos el método *setItem*:

```
1.  localStorage.setItem('key', 'value');
```

De esta forma, en nuestro espacio de almacenamiento local, tendremos un elemento llamado *key* y que tendrá por valor *value*.

RECUPERAR UN VALOR LOCALSTORAGE

Para recuperar la información utilizamos el método *getItem*:

```
1.
2.  var value = localStorage.getItem('key');
```

Este código Javascript almacena en la variable *value* el contenido almacenado para *key*.

CÓMO SABER EL NÚMERO DE ELEMENTOS ALMACENADOS EN LOCALSTORAGE

La API asociada permite el uso de *length* para conocer cuántos elementos están guardados en **localStorage.**

```
1.  alert(localStorage.length);
```

Al ejecutar este código, nos aparecería una alerta con el número de elementos almacenados en nuestro espacio local.

BORRAR UN ELEMENTO LOCALSTORAGE

Como es lógico, la API también permite eliminar un elemento que tengamos guardado. Para ello utilizamos el método *removeItem*

```
1.  localStorage.removeItem('key');
```

LIMITACIONES DEL LOCALSTORAGE

Más que limitaciones, deberíamos hablar de la gran limitación: Sólo podemos almacenar cadenas de texto. O sea, no podemos guardar booleanos (true o false), no podemos guardar arrays, objetos, floats…. sólo strings.

Estamos ante una gran limitación… pero podemos superarla con unas pocas líneas de código!! ¿Cómo? pues con ese "recurso" que cada vez cobra más fuerza: **JSON**.

Los navegadores que soportan **localStorage** (o sea, los modernos), también tienen soporte para **JSON**. Gracias a **JSON** podremos convertir un objeto (o lo que esa) en cadena de texto y almacenarlo en nuestro **localStorage**. Al mismo tiempo, con **JSON** podremos transformar la cadena recuperada de **localStorage** al objeto inicial

```
1.  // Creamos un objeto
2.  var object = { 'uno' : '1', 'dos' : '2' };
3.  // Lo guardamos en localStorage pasandolo a cadena con JSON
4.  localStorage.setItem('key', JSON.stringify(object));
5.  // Creamos una nueva variable object2 con el valor obtenido de localStorage usando
    JSON recuperar el objeto inicial
6.  var object2 = JSON.parse(localStorage.getItem('key'));
7.  // La alerta mostrará 1 por pantalla
8.  alert(object2.uno);
```

EJEMPLO PRÁCTICO DE LOCALSTORAGE

Ya sabemos, prácticamente, todo lo que necesitamos saber sobre localStorage, aunque no todo. Por ejemplo, existe un evento que permite "disparar" determinado código cuando un valor de localStorage es modificado, aunque por ahora no tiene mucha utilidad, por lo que he decidido ignorarlo

Bien, aunque los conceptos básicos están claros, vamos a crear un pequeño script que sea un "juego" de completa la frase. OJO, muy simple y muy mejorable, pero nos sirve para el caso

EJEMPLO

DESCRIPCIÓN DEL JUEGO

- El juego consiste en la primera frase de “El Quijote de la Mancha”. En la pantalla aparecerá el texto y algunas palabras estarán substituidas por ****
- El contenido de **** será editable, o sea, seleccionando el área con el ratón se podrá escribir y eliminar el texto de ese área.
- Una vez esté completada la frase, se hará clic en el botón de “Comprobar”. El script comprobará si está la frase correcta o no, avisando del resultado.
- Este script irá guardando un histórico de todos los intentos, de forma que se mostrarán, en la misma pantalla, los textos que fueron enviados a comprobación. Para almacenar estos textos utilizaremoslocalStorage. Una vez terminado el script, deberemos hacer varias pruebas, unas poniendo bien el texto y otras fallando.
- Cuando tengamos el historial bastante lleno, cerraremos el navegador. Una vez cerrado lo volveremos a abrir y comprobaremos que se mantiene el historial
- También se incorporará un botón de “Reiniciar” para eliminar nuestro historial.

CREANDO EL HTML

Como vamos a escribir nuestro código atendiendo a las buenas prácticas, nuestro código HTML no tendrá nada de CSS ni de Javascript, en su lugar se incorporarán a la cabecera las llamadas a los ficheros externos dónde tendremos el código tanto CSS como Javascript.

Así pues, lo primero es crear el documento HTML (index.html) base y la cabecera del mismo:

```
<!DOCTYPE html>
```

```
1.  <html>
2.  <head>
3.  <title>LocalStorage</title>
4.  <meta charset="utf-8" />
5.  <link rel="stylesheet" href="./style.css" />
6.  <script                                                    defer
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
7.  <script defer type="text/javascript" src="./javascript.js"></script>
8.  </head>
9.  <body>
10. </body>
11. </html>
```

- Utilizamos el doctype de HTML5.
- Tenemos nuestro title.
- Informamos que usamos utf-8 como juego de caracteres. Recordar que **el meta charset es obligatorio**.
- Cargamos el CSS gracias al tag link, indicando que es una hoja de estilos (rel stylesheet) y que el fichero a cargar es el style.css (ubicado al mismo nivel que nuestro index.html). Aquellos que estén habituados a HTML 4.01 o XHTML notarán que no se incluyo el atributo type con valor "text/css". Al igual que pasa con Javascript, HTML5 ya interpreta que se trata de un fichero de tipo text/css, no es preciso indicarlo.
- Cargamos el framework jQuery desde la URL que Google nos proporciona para ello. **Siempre es mejor usar la URL de Google que cargar el jQuery desde nuestro espacio web**.
- Por último, cargamos nuestro javascript… con el que "haremos la magia".

Bueno, ya nos toca meternos en el *body*… así que empezaremos por ponerle una cabecera con nuestro *h1*

```
1. <header>
2. <h1>Uso de LocalStorage</h1>
3. </header>
```

Empezamos con "lo divertido". Vamos a crear la zona en la que mostraremos la frase a completar y desde la cual se podrá comprobar o reiniciar.

Está claro que se trata del contenido principal de nuestra página, por lo que todo ello estará dentro de un*section*. Este *section* tendrá un *article* que contendrá todo lo que acabamos de indicar:

```
1. <section>
2. <article>
3. <fieldset>
4. <legend>Completa la frase</legend>
5. <div id="text">
6. En un lugar de la <span contenteditable="true">****</span>, de cuyo nombre no
   quiero <span contenteditable="true">****</span>, no ha mucho tiempo que vivía un
   hidalgo de los de lanza en <span contenteditable="true">****</span>, adarga
   antigua, rocín flaco y galgo <span contenteditable="true">****</span>.
7. </div>
8. <button id="check">Comprobar</button>
9. <button id="restart">Reiniciar</button>
10. </fieldset>
11. </article>
12. </section>
```

De este código, debería llamar la atención el contenido de las etiquetas *span*. Cada una de las etiquetas*span* contiene una de las palabras que deberán ser completadas. Para

permitir que se pueda escribir la palabra y eliminar los **** no vamos a utilizar "complejas" técnicas de javascript no. Vamos a utilizar el maravilloso atributo "**contenteditable**" de **HTML5** que, como es de imaginar, permite que el contenido de esa etiqueta pueda ser editado desde el navegador. Esta es otra novedad grandiosa de HTML5, muy, pero que muy buena.

Bien, ya sólo nos falta la zona dónde listaremos el historial. Se trata de información secundaria, por lo que la colocaremos dentro de un *aside*:

```
<aside>
<fieldset id="history">
<legend>Tu historial</legend>
<article></article>
</fieldset>
</aside>
```

Vemos que tenemos la etiqueta *article*, pero sin contenido dentro. Es ahí dónde pondremos el contenido que se obtenga de **localStorage**.

HTML.

```
<!DOCTYPE html>
<html>
<head>
<title>LocalStorage</title>
<meta charset="utf-8" />
<link rel="stylesheet" href="./style.css" />
<script defer src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
<script defer type="text/javascript" src="./javascript.js"></script>
</head>
<body>
<header>
<h1>Uso de LocalStorage</h1>
</header>
<section>
<article>
<fieldset>
<legend>Completa la frase</legend>
<div id="text">
En un lugar de la <span contenteditable="true">****</span>, de cuyo nombre no quiero <span contenteditable="true">****</span>, no ha mucho tiempo que vivía un hidalgo de los de lanza en <span contenteditable="true">****</span>, adarga antigua, rocín flaco y galgo <span contenteditable="true">****</span>.
</div>
<button id="check">Comprobar</button>
<button id="restart">Reiniciar</button>
</fieldset>
</article>
</section>
<aside>
<fieldset id="history">
<legend>Tu historial</legend>
<article></article>
</fieldset>
</aside>
</body>
</html>
```

APLICANDO ESTILOS CON CSS3

```
1.  * {
2.  font-family: monospace, serif;
3.  font-size: 10pt;
4.  line-height: 30pt;
5.  text-align: center;
6.  }
7.
8.  button {
9.  border: 1px solid black  ;
10. border-radius: 10px;
11. cursor: pointer;
12. margin: 20px auto;
13. padding: 0 10px;
14. }
15.
16. .correct {
17. border:2px solid green  ;
18. border-radius: 10px;
19. margin: 10px;
20. padding: 10px;
21. text-align: left;
22. }
23.
24. .error {
25. border:2px solid red  ;
26. border-radius: 10px;
27. margin: 10px;
28. padding: 10px;
29. text-align: left;
30. }
31.
32. fieldset {
33. border: 1px solid black  ;
34. border-radius: 20px;
35. box-shadow: 5px 5px 5px #888  ;
36. margin: 0 auto;
37. width: 75%;
38. }
39.
40. fieldset div {
41. text-align: left;
42. }
43.
44. legend {
45. padding: 0 10px;
46. text-transform: uppercase;
47. }
48.
49. span {
50. border: 1px solid blue  ;
51. padding: 5px;
52. }
```

Este es el código CSS que he decidido utilizar para el juego, es un código muy sencillo por lo que no requiere mayores explicaciones.

USANDO JAVASCRIPT PARA COMPROBAR EL TEXTO Y EL ALMACENAMIENTO LOCALSTORAGE

El código Javascript será el encargado de comprobar que el texto sea correcto, mostrar el historial por pantalla y guardarlo en almacenamiento local para que cuando se acceda de nuevo se vea todo el historial.

Como el código está comentado, lo pongo todo junto:

```
1.
2.  $(document).ready(function() {
3.  // Las palabras correctas por orden
4.  var values = [ 'Mancha', 'acordarme', 'astillero', 'corredor' ];
5.
6.  // Recuperamos el historial almacenado en local
7.  var history = JSON.parse(localStorage.getItem('history'));
8.
9.  // Si no existe historial, creamos un array sin contenido
10. if (history === null) {
11. history = [];
12. } else {
13. // Como existe historial, lo recorremos para mostrar por pantalla
14. // cada elemento del historial en la zona correcta del HTML.
15. $.each(history, function(key, value) {
16. $("#history").append(value);
17. });
18. }
19.
20. // Contenido que se ejecuta al clicar en "comprobar"
21. $('#check').click(function() {
22.
23. var checked = 0;
24. var right = 0;
25. var text = '';
26.
27. // Recorremos cada span del texto para compararlo con su valor.
28. $("#text span").each(function() {
29. // Si la palabra es correcta, sumamos 1 a right.
30. if ($(this).html() == values[checked]) {
31. right += 1;
32. }
33. // Sumamos 1 a la variable de "spans comprobados".
34. checked += 1;
35. });
36.
37. if (right == checked) {
38. // Si la frase es correcta, establecemos el texto a incorporar
39. // al historial dentro de una etiqueta con class "correct"
40. // para que el pintado tenga un borde verde
41. text = '<div class="correct">' + $("#text").html() + '</div>';
42. } else {
43. // Si la frase no es correcta, establecemos el texto a incorporar
44. // al historial dentro de una etiqueta con class "error"
45. // para que el pintado tenga un borde rojo.
46. text = '<div class="error">' + $("#text").html() + '</div>';
47. }
48.
49. // Incorporamos el texto al array que almacenaremos en local.
50. history.push(text);
51.
52. // Incorporamos el texto al historial por pantalla.
53. $("#history article").append(text);
```

```
54.
55. // Guardamos en local (localStorage) el array history
56. localStorage.setItem('history', JSON.stringify(history));
57.
58. if (right == checked) {
59. alert('Good Job!');
60. } else {
61. alert('You failed!');
62. }
63. });
64.
65. // Contenido que se ejecuta al clicar en "reiniciar"
66. $("#restart").click(function() {
67. // Eliminamos el elemento "history" del almacenamiento local.
68. localStorage.removeItem('history');
69. // Vaciamos el historial mostrado por pantalla.
70. $("#history article").html('');
71. });
72.
73. });
```

LO QUE VIMOS

Acabamos de ver como guardar elementos dentro de nuestro navegador, recordemos el límite que tenemos de los datos, ósea no podemos estar guardando una estructura muy compleja porque estaríamos sobrecargando al navegador, si utilizamos mucha memoria terminaríamos ralentizando nuestra aplicación ya que queda almacenado en un lugar lógico y no físico

BIBLIOGRAFIA

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API