CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com

Componentes HTML5 + JS

INDICE

1) Geolocalización

2) Drag & Drop

3) WebStorage

4) WebSQL

5) Ejemplo WebSQL

Geolocalización

- La Geolocalización es una de las características principales de HTML5, la cual empieza a tener un gran empuje en todos los ámbitos de la web, redes sociales y más.
- Veremos con solamente un poco de código cómo podremos obtener la ubicación de un usuario con HTML5, que en realidad estaremos usando la API de Geolocalización de la W3C (Consorcio World Wide Web), que es el organismo internacional que dicta lo estándares Web.
- Podremos observar que obtener la información de esta manera es aún más rápida.

Geolocalización

- Podemos obtener la posición geográfica del usuario.
- Sin embargo dado que ésto puede comprometer la privacidad del usuario, éste debe de dar siempre su permiso para poder obtenerla.
- La mayoría de los navegadores modernos soportan la geolocalización, si bien es posible que pueda haber algún fallo de disponibilidad en encontrar el sistema de localización.

Geolocalización

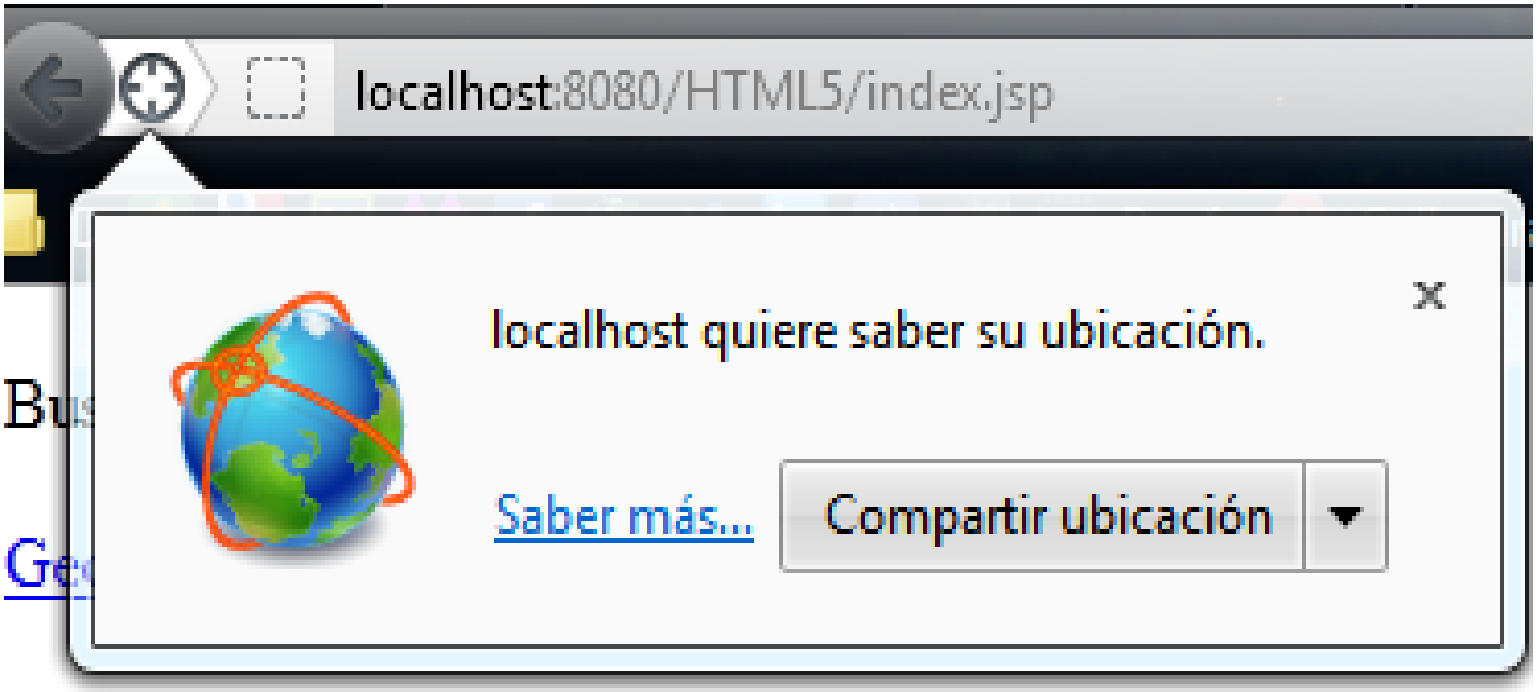
- La geolocalización da una situación mucho más exacta si el usuario se conecta con algún dispositivo con GPS, el cual será usado por el navegador para la localización.
- En caso de no haber dispositivo GPS la localización viene dada por la red de telefonía o por la configuración del ordenador.

Geolocalización

- Con el objeto **navigator.geolocation** es con el que detectamos si el navegador tiene las capacidades necesarias para detectar la Geolocalización.
- El método **navigator.geolocation.getCurrentPosition** es el que hace la labor de recuperación de la ubicación del usuario mediante el objeto positivo que se envía a la función.

Geolocalización

- Una vez que se ha llamado a este método, por medio de una función que permita su ejecución, el navegador nos preguntará si le permitimos usar la información de nuestra ubicación, lo hará de esta manera:



Geolocalización

Dicha función admite 3 tipos de parámetros (opcionales):

- **showLocation**: Funcion callback que se llamará después de obtener la localización.
- La recibiremos con un objeto de tipo Position y que nos vendrá indicada entre otras
  - position.coords.latitude: Latitud (Norte y sur)
  - position.coords.longitude: Longitud (Este y oeste)
  - position.coords.speed: Velocidad en metros por segundo
  - position.coords.altitude: Altitud en metros

Geolocalización

Dicha función admite 3 tipos de parámetros (opcionales):

- **errorHandler**: Este método será llamado en caso de que haya ocurrido algún error.
- Los tipos de errores pueden ser:
  - Unknown Error (código 0)
  - Permision Denied (código 1)
  - Position Unavailable (código 2)
  - Timeout (código 3)

Geolocalización

Dicha función admite 3 tipos de parámetros (opcionales):

- **options**: objetos con parámetros opcionales, entre ellos:
  - enableHighAccuracy: True o false, para activar que el resultado sea lo más preciso posible. Ojo con esto que puede tener efectos adversos (tiempo demasiado largo de espera, consumo de batería algo, etc)
  - timeout: Tiempo máximo en responder
  - maximumAge: Permite a los dispositivos responder inmediatamente con una posición en la caché, con un tiempo máximo indicado en milisegundos

Geolocalización

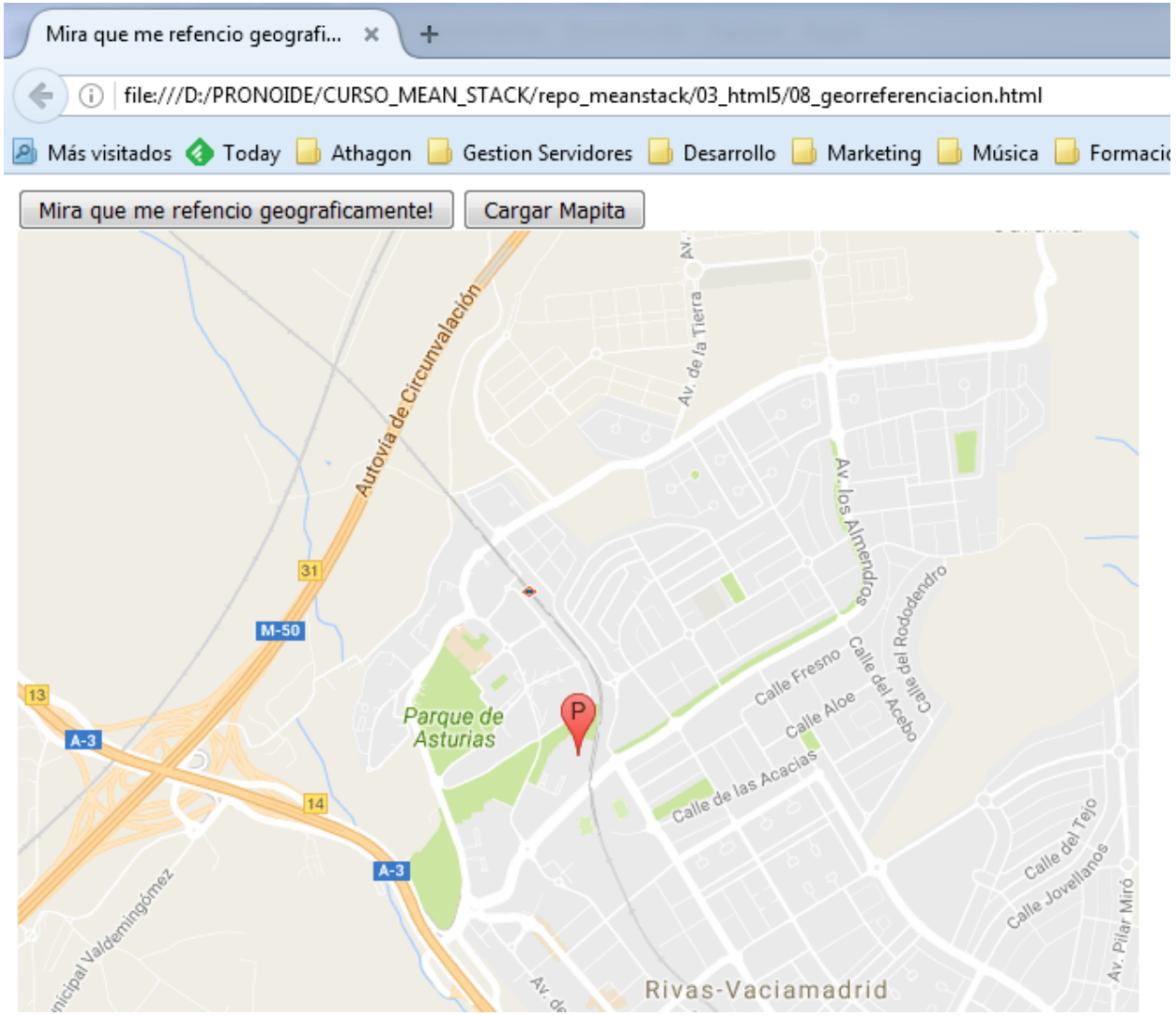
Ejemplo de llamada:

```
navigation.getCurrentPosition(
    funcionCallBack,
    funcionManejoErrores,
    {enableHighAccuracy : false, timeout : 0});

function funcionCallBack(position){
    /* Codigo */
};
function funcionManejoErrores(error){
    /* Codigo */
};
```

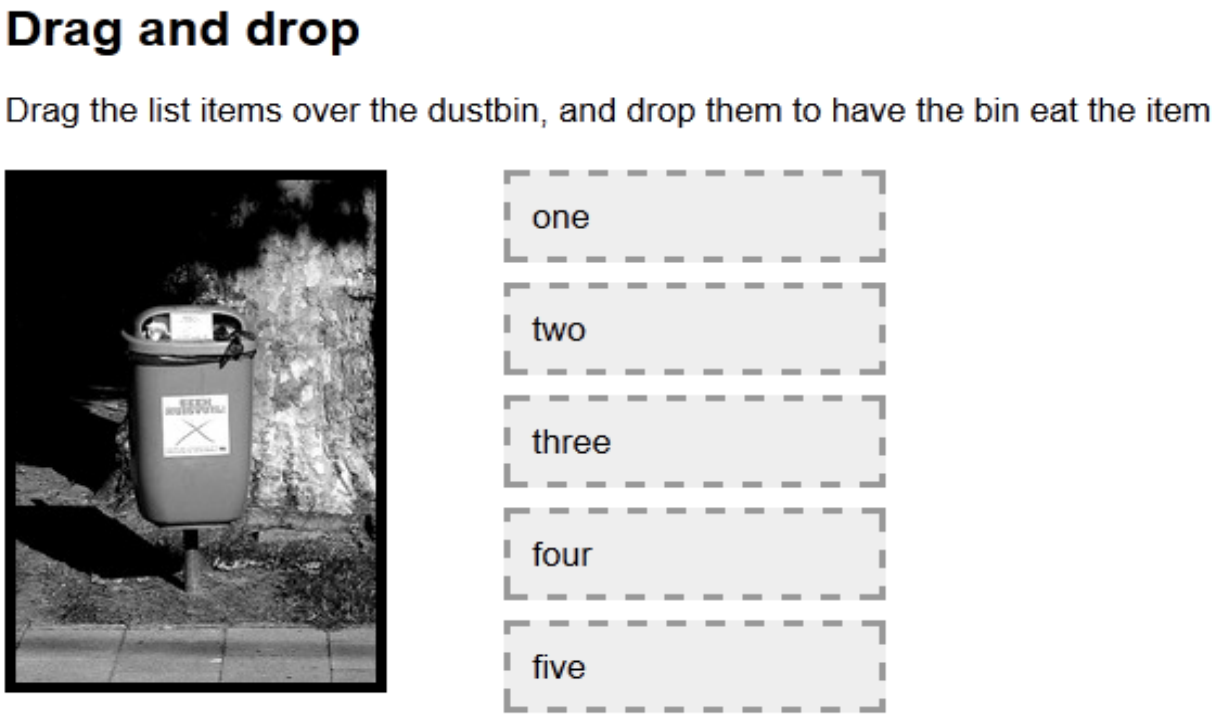
Geolocalización

Práctica: Cargar mapa de Google según la posición GPS indicada en el navegador



Drag & Drop

- Rescatado de Internet Explorer 5. Nos permite arrastrar y soltar elementos del navegador
- Crear un elemento draggable y otro dropable y ejecutamos un código para que esto ocurra:



Drag & Drop

- Para crear un elemento dropable:
  - <img src='imagenDraggable' draggable='true'>
- Eventos mas importantes asociados:
  - **ondragstart**: Se lanza justo cuando empezamos a arrastrar nuestro objeto. Es importante porque en el podremos añadir posibles valores que queramos llevar a nuestro destino mediante el datatransfer
  - **ondragover**: Se lanza cuando pasamos por una zona donde podemos soltarlo
  - **ondragenter**: Justo cuando entramos en una zona donde podemos soltarlo
  - **ondragleave**: Justo cuando abandonamos una zona donde podemos soltarlo
  - **ondrop**: Al soltar el elemento draggable sobre otro objeto. Este evento tendrá que tenerlo aquellos objetos sobre los que queramos que los elemenos dragables sean dropables

Drag & Drop

DataTransfer

- Este objeto será el que contenga los datos que se van a enviar en nuestra accion de arrastre y estará asociado a un evento
- Comúnmente, se establece en el evento ‘ondragstart’ y se lee en el evento ‘ondrop’
- Metodos:
  - **dataTransfer.setData:** Dos parámetros, el primero contendrá un string con el MIME tipe asociado, el segundo, el valor. Normalmente se utilizará el MIME ‘text/plain’ o ‘text/html’(Internet Media Type).

    Ej: event.dataTransfer.setData(‘text/plain’,’valor’)

  - **dataTransfer.getData:** Un solo parámetro, el string con el MIME tipe asociado

Ej: var valorTransferido = event.dataTransfer.getData(‘text/plain’);

Drag & Drop

DataTransfer

- Este objeto será el que contenga los datos que se van a enviar en nuestra accion de arrastre y estará asociado a un evento
- Comúnmente, se establece en el evento ‘ondragstart’ y se lee en el evento ‘ondrop’
- Metodos:
  - **dataTransfer.setData:** Dos parámetros, el primero contendrá un string con el MIME tipe asociado, el segundo, el valor. Normalmente se utilizará el MIME ‘text/plain’ o ‘text/html’(Internet Media Type).

Ej: event.dataTransfer.setData(‘text/plain’,’valor’)

  - **dataTransfer.getData:** Un solo parámetro, el string con el MIME tipe asociado

Ej: var valorTransferido = event.dataTransfer.getData(‘text/plain’);

Ver 09_draganddrop.html

Drag & Drop

Método preventDefault()

- Hay que tener en cuenta que siempre que arrastremos y arrojemos un objeto html realizará las acciones por defecto que tenga establecidas
- Por ejemplo, si hacemos una etiqueta <a> que sea draggable, cuando la arrastremos y la soltemos se ejecutará su acción por defecto que es la de abrir un enlace a la url que contenga
- Para prevenir esto, podemos hacer que evite su comportamiento por defecto usando el método preventDefault() que está asociado al evento.

WebStorage

- Web Storage es un API de HTML5 para JavaScript que permite a las páginas web **guardar información en la parte cliente**.
- Es el **sustituto natural de las tradicionales "cookies"**, aunque a diferencia de estas últimas, la información gestionada con el nuevo API sólo puede ser accedida desde el cliente, nunca desde el servidor.
- Las cookies que se utilizan actualmente siempre **se mandan como una cabecera HTTP en cada conexión** entre cliente y servidor, para que ambos vean la misma información.

WebStorage

- El problema es que esto implica un sobrecoste en la comunicación forzando a que el tamaño de los datos almacenados sea normalmente muy pequeño.
- Además de que **permiten realizar ataques por parte de malintencionados** que pueden cambiar "al vuelo" el contenido de las cookies intercambiadas entre clientes y servidores.

WebStorage

- Y sobre todo, pueden dar lugar a inconsistencias entre el estado de un cliente web y la información contenida en una cookie, por ejemplo cuando se abren dos ventanas de una misma web, o cuando en una web de compras se añade un artículo nuevo a un carrito de la compra gestionado con una cookie y luego se pulsa el botón de retroceso del navegador.

WebStorage

- El nuevo API, al no intercambiar información con el servidor, permite que la cantidad de datos almacenados en el cliente pueda ser mucho mayor, del orden de 5 Mb según la recomendación, totalmente arbitraria, que aparece en la documentación oficial, y que en la práctica dependerá de la política de gestión de cuotas de cada navegador en particular.

WebStorage

- El API define dos tipos de almacenamiento, uno de ellos temporal y a nivel de ventana (sessionStorage), y el otro persistente y a nivel de web (localStorage):

WebStorage

- **sessionStorage** es por sesión y ventana, de forma que dos pestañas del navegador abiertas al mismo tiempo para una misma web pueden tener información distinta según lo que se esté haciendo en cada momento en cada una de ellas.
- Al cerrar la sesión se pierde la información

WebStorage

- Los datos se almacenan de forma estructurada en forma de pares (clave, valor).
- Donde la clave es una cadena de texto y valor puede ser de cualquier tipo, aunque en la práctica **sólo se permite String** para los valores.
- En la documentación oficial se indica además que los navegadores deberían elevar una excepción en caso de que se intentara almacenar un objeto de tipo ImageData.

WebStorage

- El API de JavaScript define una interface "Storage" bastante sencilla e intuitiva.
- Si se quiere utilizar almacenamiento temporal se utiliza "sessionStorage", y si se quiere almacenamiento permanente "localStorage".
- Estos atributos están **directamente disponibles a nivel de "window"** y no es necesario instanciarlos.
- Para los ejemplos se usará "localStorage".

WebStorage

METODOS:

- Con el método "setItem(key, value)" se almacena un valor para una clave dada.
- Si ya existe un valor previo almacenado para la clave dada se sobreescribe con el nuevo.

```
localStorage.setItem("user", "Holmes");
```

WebStorage

METODOS:

- Con el método "geItem(key)" se recupera el valor almacenado para una clave dada.
- Si no existe valor para la clave dada el método retorna null.

```
localStorage.getItem("user");
```

WebStorage

METODOS:

- Con el método "removeItem(key)" se borra una clave dada y su valor asociado del almacén.
- Si la clave dada no existe no se devuelve ningún error.

```
localStorage.removeItem("user");
```

WebStorage

METODOS:

- Con el método "key(index)" se obtiene el nombre de la clave para un índice dado.
- Este método puede resultar un poco extraño de entrada, pero es útil para recorrer todas las claves que actualmente se encuentran almacenadas.

```
localStorage.key(i)
```

WebStorage

METODOS:

- Si en el almacén hay N claves, entonces se pueden recuperar sus nombres iterando entre 0 y N-1.

```
for (var i= 0; i < localStorage.length; ++ i){
  hacerAlgoCon( localStorage.key(i) ); }
```

WebStorage

METODOS:

- Finalmente, comentar que si lo que se quiere almacenar son objetos más complejos de JavaScript, siempre se puede utilizar las utilidades de conversión entre objetos JSON y String.

```
var user = {
name: "Holmes", friend: "Watson"};
...
localStorage.setItem("user", JSON.stringify(user) );
… user = JSON.parse( localStorage.getItem("user") );
```

WebStorage

- En la especificación oficial se indica además que cualquier tipo de cambio en el almacén debe disparar un evento de tipo "StorageEvent", de forma que cualquier ventana con acceso al almacén pueda responder al mismo.

```
window.addEventListener("storage", onStorage, false);
function onStorage(event){
//Evento recibido de tipo StorageEvent
}
```

WebStorage

- El evento tiene los atributos "key", "oldValue", "newValue", "storageArea" y "url", para indicar la clave afectada, el valor anterior, el nuevo valor, el almacén afectado, y la url desde la que se realizó el cambio.
- Si la clave es nueva el valor anterior llega como null.
- Si la clave está siendo borrada el nuevo valor llega como null.
- Si el almacén entero está siendo borrado la clave y los dos valores llegan a null.

WebStorage

Ejercicio: Formulario para guardar nombre y apellidos en los datos del navegador local (localStorage), y dos etiquetas con botón de cargar donde se recuperarán los datos:

WebSQL

- Otra de las grandes novedades de HTML5 es las nuevas alternativas respecto a la forma de guardar información en el cliente por parte de una web.
- Hasta ahora, la única alternativa que se tenia desde una web era usar cookies.
- HTML5 introduce tres nuevas formas, “local” y “session” storage y por otro lado la posibilidad de manejar toda una **base de datos relacional** que reside en cada cliente.

WebSQL

- A diferencia del localstorage o sessionstorage, que son mas similares a las cookies, el hecho de manejar una **base de datos totalmente funcional** es una novedad bastante importante que puede ayudar a nuestras webs a guardar información de una forma mucho mas estructurada.

WebSQL

- Como hemos dicho, la base de datos es de tipo relacional, por lo que el lenguaje para “hablar” con ella será enteramente SQL.
- Aunque expresamente la **W3 desaconseja** ligar el estándar a una implementación concreta, en la practica, prácticamente la **totalidad de navegadores han usado SQLlite** para implementar la gestión de la base de datos.

WebSQL

- Como sabemos, no el SQL que entiende todas las bases de datos es igual, existen lo que se denominan “dialectos”, debido a que SQL lite es casi la opción “de-facto” podremos usar el **“dialecto” del SQLite** en las sentencias.
- Aunque, al igual que el estándar desaconsejaba ligarse a una implementación, no es buena idea ligar el código a un dialecto, pues puede ser que en algún momento uno de los navegadores decida utilizar otro “backend” de bases de datos, y haga nuestro código incompatible.
- Lo ideal es usar, en la medida de lo posible, el **SQL mas cercano al estándar**.

WebSQL

- Pese a todo esto, de momento y hasta que no se plantee una implementación independiente, el estándar de HTML5 respecto al “WebSQL” apunta a que el dialecto a seguir es del SQLite.
- Pasando ya a como usar la base de datos desde javascript, el primer paso es obtener una referencia a la base de datos para poder operar con ella.
- Para ello usaremos la función **openDatabase()** donde le pasaremos, el **nombre** de la base de datos, la **versión** del esquema y el **tamaño** estimado.

WebSQL

- Una vez tenemos la referencia de la base de datos, vamos a ver que, prácticamente la **totalidad de los métodos son asíncronos**, por lo que típicamente los argumentos serán **callbacks**.
- En este caso Javascript sale a nuestra ayuda, como seguramente sabremos, en Javascript podemos encapsular en una variable la declaración de una función, por lo que la forma mas común de pasar los parámetros a las funciones que operan en la base de datos es incluir la declaración de los propios callbacks, ya que **habitualmente, no será muy útil reutilizar dichos callbacks**.

WebSQL

- Las dos funciones más usadas sin duda van a ser **“transaction” y “executeSql”**
- Con la primera crearemos una transacción que se ejecutará contra la base de datos de forma atómica (bueno, esa es la definición de transacción, no?)
- Con la segunda, como su nombre indica ejecutaremos las propias sentencias sql.
- A continuación veremos un simple ejemplo en el que creamos una base de datos.

Ejemplo WebSQL

```
var webdb = {};
webdb.db = null;

// Función para crear la base de datos
webdb.open = function(options) {
    if (typeof openDatabase == "undefined") return;

    // Opciones por defecto
    var options = options || {};
    options.name = options.name || 'noname';
    options.mb = options.mb || 5;
    options.description = options.description || 'no description';
    options.version = options.version || '1.0';

    // Definimos el tamaño en MB
    var dbSize = options.mb * 1024 * 1024;

    // Cargamos la base de datos
    webdb.db = openDatabase(options.name, options.version,
                            options.description, dbSize);
}
```

Ejemplo WebSQL

```
// ExecuteSql
webdb.executeSql = function(sql, data, onSuccess, onError){
    if (!webdb.db) return;
    webdb.db.transaction(
        function(tx){
            tx.executeSql(sql, data,onSuccess,onError);
    });
}
// Ejemplo
var opt = {
    name: "ejemplo",
    mb: 1,
    description: "Base de datos de ejemplo",
    version: "1.0"
};
```

Ejemplo WebSQL

```
// Abrimos la base de datos
webdb.open(opt);

// Creamos la tabla
webdeb.executeSql('CREATE TABLE IF NOT EXISTS ejemplo (ID INTEGER
PRIMARY KEY ASC, texto TEXT, added_on DATETIME"', [],
            function(tx, r){
                alert("Tabla creada");
            },
            function(tx, e){
                alert("Se ha producido un error: "e.message);
            });

// Insertamos un nuevo elemento
webdb.executeSql('INSERT INTO ejemplo (texto, added_on) VALUES (?,?)',
['Mensaje de ejemplo', new Date()],
            function(tx, r){
                alert("Elemento introducido");
            },
            function(tx, e){
                alert("Se ha producido un error: "e.message);
            });
```