

8

Almacenamiento de datos en el lado cliente

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer los mecanismos de almacenamiento web del lado del cliente.
- ✓ Comentar la especificación web Storage de la W3C.
- ✓ Repasar y comparar las diferentes tecnologías y sus implantaciones: los objetos de almacenamiento web de HTML 5 e IndexedDB.
- ✓ Profundizar en los conceptos genéricos de las bases de datos del lado del cliente.

En este capítulo presentamos los conceptos básicos del almacenamiento de datos e información del lado del cliente. Estudiaremos las desventajas de las *cookies*, como una de las tecnologías más utilizadas en el pasado y que actualmente sigue manteniendo su vigencia. Así mismo, profundizaremos en los nuevos mecanismos introducidos por HTML 5 y su implantación en los navegadores actuales. Por último, comentaremos las tendencias actuales con las bases de datos del lado del cliente y el almacenamiento orientado a objetos JavaScript.

8.1 ALMACENAMIENTO WEB

Desde 1989 las páginas web han surgido como herramientas para visualizar e intercambiar información. Con el paso del tiempo han evolucionado en aplicaciones web, permitiendo interactividad y personalización (Berners-Lee, 1999).

Uno de los pilares de la personalización se encuentra en el concepto de sesión y en la habilidad de almacenar datos del usuario que utiliza el sitio web en el mismo navegador cliente con el fin de mejorar la experiencia del usuario. Una de las premisas en ese sentido es la siguiente: si la información pertenece al usuario debe estar en la localización del usuario (es decir en el navegador cliente). A continuación veremos las diferentes opciones de almacenamiento de datos en los navegadores clientes. La opción inicial son las *cookies* (Barth, 2011).

8.1.1 LAS COOKIES

El protocolo de transferencia de hipertexto o *Hypertext Transfer Protocol* (HTTP) es el mecanismo utilizado para el intercambio de información en Internet. Pero HTTP es un protocolo sin estado, por tanto, el servidor web administrará cada nueva petición HTTP de manera independiente y sin retener ningún tipo de información, como, por ejemplo, los valores de las variables y campos que se han utilizado en cada solicitud. Cuando un cliente solicita una página web, el servidor entrega la página y, a continuación, olvida todo sobre la petición y el cliente.

Por supuesto, esto disminuye la capacidad de las aplicaciones web por lo que la construcción de aplicaciones que necesiten mantener el estado entre peticiones es una tarea más complicada. Es por ello que, teniendo en cuenta esta necesidad, se implementan mecanismos para lograr la persistencia de los datos, especialmente a partir de las sesiones.

Una sesión es una técnica para vincular datos de un usuario identificado a lo largo de los distintos accesos a una aplicación web. Entre los diferentes usos están el de mejorar la usabilidad y la experiencia del usuario. Un ejemplo muy común es el de mantener al usuario registrado en una página que le ha pedido identificación (*login* y contraseña).

Uno de los mecanismos más utilizados para mantener información en el cliente son las *cookies* (Barth, 2011): una pequeña cantidad de datos almacenada en el cliente en forma de pares clave/valor, con una fecha de expiración y relacionada con un dominio determinado (en realidad contiene más atributos pero de forma genérica estos son los más importantes).

La fecha de expiración es un parámetro opcional que indica el tiempo que se conservará la *cookie*. Si no especificamos la fecha de expiración la *cookie* se eliminará cuando el usuario cierre todas las instancias del navegador. Así mismo, el nombre de dominio total o parcial funcionará como filtro. El navegador enviará la *cookie* al servidor cuyo nombre de dominio sea igual a ese atributo. Por ejemplo, si especificamos el nombre de dominio de la forma “.google.es” el

navegador incorporará la *cookie* a *www.google.es*, *maps.google.es*, *scholar.google.es*, etc. En caso de no indicar este atributo realizamos el filtrado más restrictivo posible y la *cookie* será devuelta solo al servidor que la ha originado.

La información contenida en la *cookie* será enviada junto con cada petición HTTP al servidor. Esto tiene como ventaja evidente el acceso inmediato por parte del servidor a los datos del cliente. Tiene como principal desventaja la disminución del rendimiento y las limitaciones de la cantidad de datos a ser almacenado. Así mismo, las *cookies* son almacenadas en un fichero, usualmente en el espacio destinado al navegador en el disco duro del ordenador en texto plano. Este fichero se leerá al abrir nuevamente el navegador y las *cookies* se gestionarán en memoria hasta que el navegador se cierre. En ese momento se grabarán aquellas *cookies* que no hayan expirado. También existen otras limitaciones, como la cantidad de *cookies* asociadas a un dominio determinado o el tamaño máximo (el cual suele ser de 4096 bytes).

Respecto a la seguridad, las *cookies* usualmente persisten en texto plano por lo que es muy difícil asegurar que los datos no sean corruptos o evitar que información sensible del usuario pueda hacerse pública.

8.1.2 PROBLEMAS CON LAS COOKIES

Aunque es una de las tecnologías más extendidas, las *cookies* tienen una serie de inconvenientes (Hope, 2008). El principal inconveniente tiene que ver con que los usuarios las relacionan con aspectos maliciosos. En realidad, una *cookie* es solo información en texto plano administrable por el mismo usuario y en ningún caso es código fuente interpretable.

Parte de la sencillez de las *cookies* genera inconvenientes:

- Cada navegador tendrá sus propias *cookies*.
- Las *cookies* no diferencian entre usuarios que utilicen el mismo navegador en una misma sesión del sistema operativo. Muchas veces queda almacenada la información de nuestra tarjeta de crédito al realizar una transferencia bancaria.
- Son vulnerables a los *sniffer* (programas que pueden leer el contenido de peticiones y respuestas HTTP) debido a que estas se realizan en texto plano.
- Las *cookies* pueden ser modificadas en el cliente, lo cual podría aprovechar vulnerabilidades del servidor. Si, por ejemplo, una *cookie* contiene información sobre una compra vía web, cambiando el precio de compra el servidor podría permitir al atacante pagar menos. Esto es fácil de resolver, indicando en la *cookie* solo identificadores de sesiones en el servidor y no directamente información sensible.

8.1.3 LAS COOKIES DE FLASH

Uno de los avances más significativos en las páginas web ha sido la inclusión de Flash. Esta tecnología es utilizada para crear y manipular gráficos vectoriales logrando animaciones y sitios web interactivos (Keefe, 2008). Podemos encontrar aplicaciones web realizadas en un gran porcentaje con Flash e información sobre sus ventajas respecto de HTML. Aun así, actualmente en sitios que no son comerciales se utiliza la tecnología Flash solo en publicidad y en visores de vídeos.

La tecnología Flash también utiliza almacenamiento del lado del cliente, con los denominados *Local Shared Object* (Objeto Local Compartido) o como comúnmente se los llama: *cookie flash*.

Son utilizados con el mismo propósito que las *cookies*: para almacenar en el cliente pequeñas cantidades de información que serán accedidas en sesiones posteriores. La diferencia inicial se encuentra en su gestión, ya que los navegadores por lo general no implementan un sistema para su modificación y borrado. Otra diferencia es su peso (hasta 100 kilobytes, en lugar de los 4 kilobytes de las *cookies* normales).

Adobe ofrece desde su página un programa en Flash que nos permite gestionar las *cookies flash*, visualizarlas y borrarlas. Así mismo, en el panel de configuración global de almacenamiento podemos controlar cuánto espacio de disco utilizan los sitios web para almacenar información o no permitir el almacenamiento de información en el equipo.

8.1.4 LA ESPECIFICACIÓN WEB STORAGE DE LA W3C

El almacenamiento web ha cobrado más fuerza con la nueva especificación de HTML 5. Ha sido estandarizado por el *World Wide Web Consortium* (W3C) y actualmente se encuentra en una especificación por separado siendo implementado por Internet Explorer 8 y Firefox entre otros navegadores.

HTML 5 incluye dos nuevos objetos para el almacenamiento de datos en el cliente: los `sessionStorage` y los `localStorage` (W3C, 2011a). Más adelante detallaremos el uso de estos objetos. En la Tabla 8.1 podemos ver una lista de navegadores que soportan esta tecnología.

Tabla 8.1 Listado de navegadores que implementan Web Storage

Característica	Internet Explorer	Firefox	Chrome	Opera	Safari
<code>localStorage</code>	8	3.5	4	10.50	4
<code>sessionStorage</code>	8	2	5	10.50	4

Esta especificación introduce dos mecanismos relacionados para obtener la persistencia de datos de manera estructurada del lado del cliente, similares al mecanismo de las *cookies*. Una de las principales diferencias está en que el contenido de las *cookies* es enviada al servidor en cada petición. En HTML 5 no, de hecho la información solo podrá ser accedida desde el lado del cliente por lo que es posible almacenar gran cantidad de información sin afectar el rendimiento de la aplicación web. Por último indicar que se utiliza JavaScript para realizar el almacenamiento y acceso a los datos.

Algunos escenarios de implementación y uso son los siguientes:

- **Almacenamiento de datos.** Los datos son almacenados en el cliente y pueden ser pasados al servidor por intervalos, en lugar de en tiempo real. Aclaración: utilizando `localStorage` los datos también estarán disponibles entre peticiones y sesiones del navegador.
- **Utilización fuera de línea.** Como consecuencia de que la relación entre los datos almacenados y el navegador no se pierden entre sesiones (para el `localStorage`).

- **Mejora de rendimiento.** Podemos almacenar datos estáticos (por ejemplo imágenes en codificación Base64) que no serán nuevamente obtenidos mediante peticiones al servidor.
- En el caso del `sessionStorage` no existe relación entre lo almacenado en diferentes pestañas o ventanas de un mismo navegador.
- Con el objeto `sessionStorage` existe la seguridad de que los datos serán borrados una vez termine la sesión de la ventana que lo ha utilizado.

SessionStorage

Actualmente podemos encontrarnos con un problema para realizar múltiples transacciones en diferentes ventanas o pestañas de un navegador si lo hacemos al mismo tiempo, en caso de que la aplicación utilice *cookies* para mantener el estado. Las diferentes instancias del navegador (ya sea en diferentes pestañas de una misma ventana o en diferentes ventanas) obtendrán información de una misma *cookie* debido a que todas las ventanas están asociadas a una misma sesión (ver Figura 8.1).

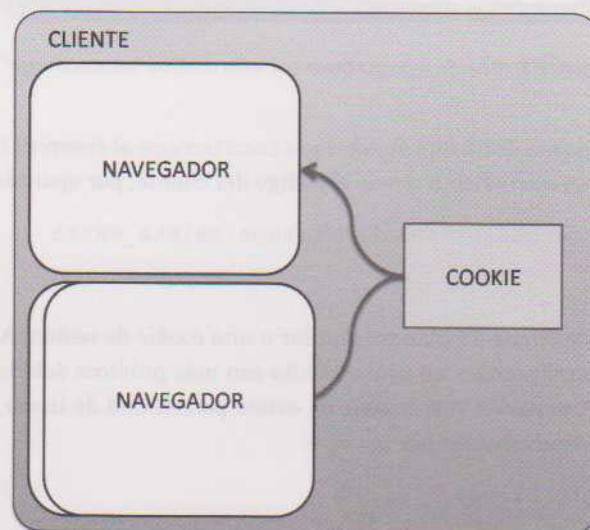


Figura 8.1. Relación entre ventanas que usan cookies

La especificación actual de *Web Storage* incluye un nuevo objeto llamado `sessionStorage`. Las aplicaciones web pueden agregar información a este atributo el cual estará accesible durante toda la sesión. El objeto `sessionStorage` se instancia por sesión y ventana, por lo que dos pestañas del navegador abiertas al mismo tiempo y para un mismo sitio web pueden tener información distinta (ver Figura 8.2). Al cerrar la sesión se pierde la información.

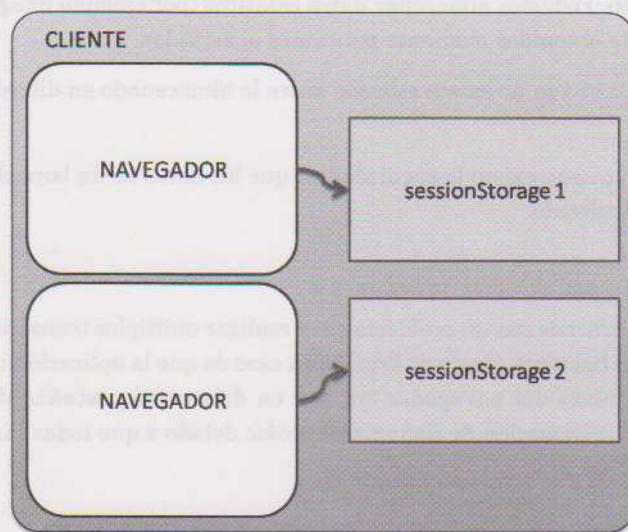


Figura 8.2. Relación entre ventanas que usan el objeto "sessionStorage"

A continuación veremos un ejemplo de uso del objeto sessionStorage al reservar un vuelo y poder elegir maletas extras. La utilización de este objeto se realizará desde el código del cliente, por ejemplo:

```
<input type="checkbox" onchange="sessionStorage.maleta_extra
= checked ? 'true' : ''>
```

Al parecer este tipo de objeto funciona de manera similar a una *cookie* de sesión. Aun así, existen diferencias. En cuanto a la seguridad, los datos almacenados en este atributo son más públicos debido a que no existe la posibilidad de esconder información entre los usuarios. Así mismo, no existe posibilidad de iterar entre los items y es necesario pedir el valor del par clave/valor directamente por la clave.

```
if (sessionStorage.maleta_extra) { ... }
```

En cuanto al tiempo de expiración, no es posible indicarlo. Esto podría parecer una desventaja en relación con las *cookies* de sesión. En otro sentido, una de las mayores ventajas teóricas de este objeto es su persistencia entre caídas del cliente.

A continuación indicamos los métodos del objeto sessionStorage:

- **setItem()**. Para agregar un nuevo par clave/valor, por ejemplo una maleta adicional al comprar un billete de avión utilizamos el método `setItem()` con la clave "maleta" y el valor "1":

```
sessionStorage.setItem("maleta", "1");
```

- **getItem()**. Para obtener el dato almacenado utilizamos el método `getItem()` con el nombre de la clave:

```
var item = sessionStorage.getItem("maleta");
```

- **removeItem()**. Para eliminar el par clave/valor existe el método `removeItem()` que deberá ser llamado con la clave o con la posición del elemento a eliminar:

```
var item = sessionStorage.removeItem("maleta");
var item = sessionStorage.removeItem(1);
```

- **clear()**. Este método borra todos los elementos de la lista `sessionStorage`.

```
sessionStorage.clear();
```

- Así mismo, podemos utilizar otros atributos, por ejemplo el atributo `length` para conocer la cantidad de elementos clave/valor almacenados:

```
var cantidad_elementos = sessionStorage.length;
```

Como ejemplo de utilización podemos ver un código JavaScript para mantener la información de un campo de texto de búsqueda, después de que el navegador sea refrescado accidentalmente.

El objetivo del código que veremos a continuación es mantener el contenido del elemento “busqueda” en el objeto `sessionStorage`. Para utilizamos la función `setInterval()` de JavaScript, la cual tiene como primer parámetro el nombre de una función que se ejecutará cada cierto intervalo. Como segundo parámetro pasamos el intervalo de ejecución en milisegundos. Por ejemplo, a continuación se ejecutará cada segundo la función “refrescar”.

```
setInterval(refrescar,1000);
```

Volviendo con el ejemplo, lo primero que necesitamos es obtener el elemento que queremos persistir:

```
var busqueda = document.getElementById("busqueda");
```

A continuación detallamos la función “refrescar”, la cual guarda el valor de la variable “búsqueda”. Recordando que esta función se ejecutará cada segundo, en todo momento la tendremos en el objeto `sessionStorage` el elemento de nombre “autoguardado” conteniendo el valor de la variable “búsqueda”.

```
refrescar(){
    sessionStorage.setItem("autoguardado", busqueda.value);
}
```

Recapitulando el ejemplo, la primera vez que se interpreta el código de la página web se carga la variable “búsqueda” y cada segundo se refresca el elemento “autoguardado” con el valor de la variable.

En caso de perder el contenido de la variable “busqueda” y si existe el elemento “autoguardado” podremos recuperar su valor:

```
if ( sessionStorage.getItem("autosave")){
    search.value = sessionStorage.getItem("guardado");
}
```

El código completo es el siguiente:

```
var busqueda = document.getElementById("busqueda");

if ( sessionStorage.getItem("autoguardado") ) {
    busqueda.value = sessionStorage.getItem("guardado");
}

setInterval(refrescar,1000);

refrescar(){
    sessionStorage.setItem("autoguardado", busqueda.value);
}
```

Para terminar, este código fuente lo podemos asociar a un evento del campo de texto “busqueda”, para que se ejecute cada vez que el contenido del texto cambie.

LocalStorage

El segundo mecanismo de almacenamiento está diseñado para los datos que se extienden a lo largo de múltiples ventanas y múltiples sesiones. Está orientado a las aplicaciones que necesitan mantener gran cantidad de información del usuario (en el orden de los megabytes), principalmente por motivos de rendimiento.

Para ello, la W3C ha definido el objeto `localStorage`. De esta manera podemos acceder al área de almacenamiento local del dominio. Puede ser accedido cada vez que se visita el dominio (los subdominios no son válidos) y todas las sesiones abiertas sobre la misma web ven la misma información (ver Figura 8.3).

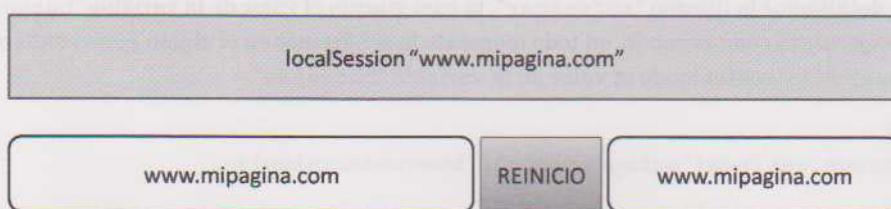


Figura 8.3. Persistencia del objeto `localStorage`

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. Esto ocurrirá ante cambios como: agregado o modificación de un elemento, borrado de un elemento o de todos ellos.

Las cuatro propiedades del evento las podemos ver en la Tabla 8.2.

Tabla 8.2 Propiedades del evento storage

Propiedades	Internet Explorer
url	El dominio asociado con el objeto que ha cambiado.
storageArea	Representa el objeto localStorage o sessionStorage afectado.
key	La clave del par clave/valor que ha sido agregado, modificado o borrado.
newValue	El nuevo valor asociado con la clave. Será null si trata de una eliminación.
oldValue	El antiguo valor.

A continuación explicamos el ejemplo clásico de contar la cantidad de veces que se ha visitado una página. Debido a que con el objeto localStorage no se pierde entre sesiones de un mismo dominio podemos realizar un contador de la siguiente manera:

```
<p>
    Ud. ha visto esta página:
    <span id="cuenta"> muchas </span>
    veces.
</p>

<script>
    if(!localStorage.cuenta)
        localStorage.cuenta = 0;

    localStorage.cuenta = parseInt(localStorage.cuenta) + 1;
    document.getElementById("cuenta").textContent =
        localStorage.cuenta;
</script>
```

En este caso inicializamos el elemento “cuenta” con el valor 0. Como veremos en el siguiente código, algunos navegadores permiten realizar el acceso directo a las claves del objeto como si se tratases de una variable. Es decir, en lugar de incluir la sentencia:

```
localStorage.setItem("cuenta","0");
```

Realizamos lo siguiente:

```
localStorage.cuenta = 0;
```

Otro aspecto importante aquí es que JavaScript realiza la conversión automática de tipo de dato numérico a cadena de caracteres. Por ello, al recuperar el valor almacenado es necesario transformarlo en tipo numérico para que sea tratado como un contador y sumarle la nueva entrada:

```
localStorage.cuenta = parseInt(localStorage.cuenta) + 1;
```

La persistencia de los datos

Una de las características importantes de este tipo de objetos es el tener una capacidad mayor de almacenamiento. Dependiendo el navegador, existirán diferentes estrategias de implantación en cuanto la forma de escritura de los datos. Firefox, por ejemplo, realiza el almacenamiento de manera síncrona, es decir que persiste inmediatamente los datos que se almacenan mediante código. En cambio en Internet Explorer los datos se escriben de manera asíncrona y, por tanto, puede existir una diferencia (mayor cuanto mayor sean los datos) entre el tiempo de asignación y el tiempo de escritura. La principal ventaja de ello es la rapidez en la ejecución del código, aunque existe mayor peligro de perder los datos.

En Internet Explorer 8 podemos forzar la persistencia mediante los métodos `begin()` y `commit()` (MSDN, 2009). Las asignaciones de datos que se encuentren encerrados entre la llamada de ambos métodos se realizarán de manera síncrona. Por ejemplo:

```
sessionStorage.begin();  
  
sessionStorage.setItem("figural1", "R0lGODlhNQAkAKIAAHJycvwHB  
wAAqwcjC+3PDv///wAAAAAACwAAAAANQAKAA");  
  
sessionStorage.commit();
```

ACTIVIDADES 8.1



- Investigue el uso del objeto `globalStorage` y su relación con los objetos `localStorage` y `sessionStorage`.

8.2 BASES DE DATOS SQL (STANDARD QUERY LANGUAGE) EN ENTORNO CLIENTE

En HTML 5 se especifican las características de un sistema de base de datos de objeto en el lado del cliente. A continuación explicaremos los conceptos centrales de esta nueva funcionalidad.

Por un lado es necesario aclarar lo que se considera una base de datos. En ese sentido lo podríamos definir como un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Esta misma definición puede dársele a las *cookies* o los objetos `sessionStorage` o `localStorage` (comúnmente llamados

DOM Storage). La diferencia principal entre estos últimos y una base de datos es la estructuración de los datos, lo que permite mantener una mayor eficiencia de los mismos y la aplicación de búsquedas directas.

Por otro lado, es necesario explicar la ventaja de tener una base de datos local en lugar de utilizar una base de datos en el servidor. Las bases de datos locales permiten almacenar datos en el cliente teniendo acceso a esta información sin necesidad de estar en línea.

Dentro de las bases de datos más utilizadas encontramos dos: las bases de datos relacionales y las bases de datos orientadas a objeto. La idea fundamental en la base de datos relacional es el uso de las relaciones entre conjunto de tuplas (comúnmente representadas por tablas).

Aunque las bases de datos relacionales son actualmente las más utilizadas, tienen una desventaja respecto de los paradigmas de programación actuales. Las aplicaciones se encuentran mayormente desarrolladas a partir de la orientación a objetos, donde se programan objetos compuestos por datos, comportamiento y relacionados con otros objetos. Sin embargo, las bases de datos actuales no están preparadas para almacenar objetos sino para almacenar tuplas (también llamados registros). En cambio en las bases de datos orientadas a objetos se realiza la persistencia de los objetos utilizados en la programación de manera transparente, sin necesidad de transformar los datos (ver Figura 8.4).



Figura 8.4. Persistencia en bases de datos relacionales y orientadas a objetos

Actualmente en las aplicaciones web existen principalmente dos técnicas de bases de datos locales:

- **WebSQL (basado en SQLite)**. Actualmente no tiene más soporte pero ha sido y es utilizado por los navegadores.
- **Indexed Database API**. Impulsado por la W3C y Oracle es la propuesta de estándar actualmente vigente. Se encuentra parcialmente implementado en las últimas versiones de los navegadores.

8.2.1 WEBSQL

Esta propuesta ya no se encuentra en mantenimiento activo por el grupo de trabajo de aplicaciones web de la W3C desde noviembre de 2010 (W3C, 2010). Está basada en SQLite, un gestor de base de datos relacionales construido en el lenguaje C.

Entre las ventajas relativas que tiene este tipo de motor de base de datos está que se enlaza directamente con la lógica de negocio de la aplicación (en este caso, por ejemplo con el código del lado del cliente). Por lo tanto, la base de datos se encuentra en el cliente y no en el servidor (ver Figura 8.5).

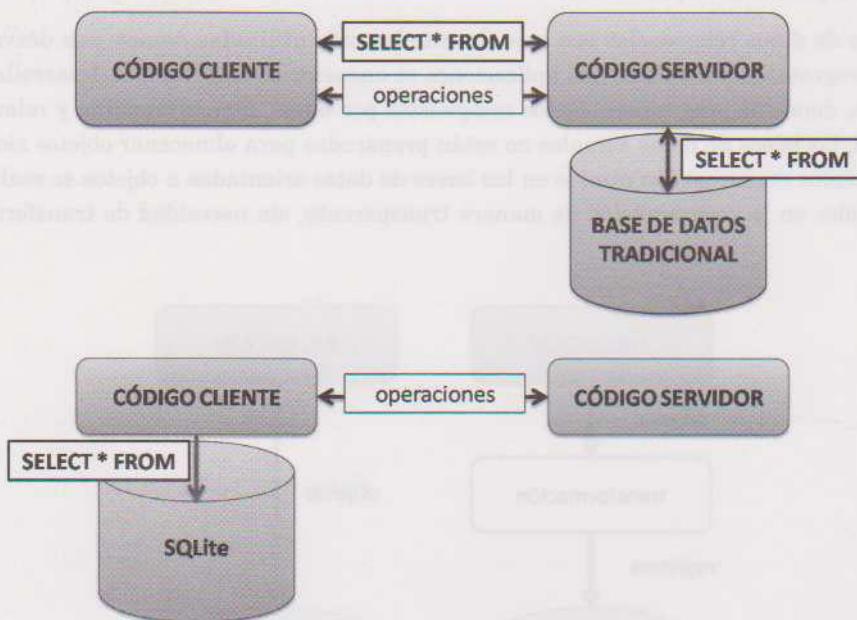


Figura 8.5. La base de datos del lado del servidor y del lado del cliente

La API de esta propuesta contiene principalmente tres métodos que consisten en la apertura y creación de la base de datos, la creación de las transacciones (con las que se asegura el éxito de la operación con la base de datos) y la ejecución o aplicación de consultas SQL.

Actualmente esta propuesta no es estable y los principales navegadores del mercado (Microsoft y Mozilla) no lo soportan, apostando por el *Indexed Database API*.

8.2.2 INDEXED DATABASE API

Esta propuesta, también denominada IndexedDB es una interfaz de aplicación de programas (API) para el almacenamiento de gran cantidad de datos de manera estructurada utilizado en HTML 5 y es el sustituto natural de WebSQL cuya especificación ha sido abonada en el año 2010 por la W3C (W3C, 2011b).

Definiciones

Debido a que IndexedDB almacena directamente objetos es posible persistir objetos JavaScript. Para ello hace uso de un mecanismo denominado *Object Store* para lograr la persistencia (ver Tabla 8.3 para una comparación entre conceptos relativos y esta API).

Tabla 8.3 Comparación entre conceptos de bases de datos relacionales y la tecnología Indexed Database API

Característica	BD relacional	Indexed Database API
Tabla	La tabla contiene filas y columnas.	Los <i>Object Store</i> o contenedores que contienen objetos Javascripts y claves.
Consultas	SQL.	Cursores y rangos de clave. Son necesarios los índices para realizar las búsquedas.
Transacciones	De lectura/escritura y a nivel de tablas o registros.	<code>Read_Version_Change, Read_Write, Read_Only.</code>
Estructura de datos	Cumple con las reglas de forma normal.	Estructura no normalizada.

En IndexedDB no se trabaja con el concepto de consultas SQL, en cambio se trabaja con índices y cursores que recorren los objetos. El *Object Store*, que a partir de ahora llamaremos “contenedor” almacena pares clave/valor. Este concepto de pares clave/valor es utilizado también en los objetos *DOM Storage* vistos anteriormente. La diferencia se encuentra en que, en este caso, los valores por lo general serán objetos con una estructura compleja. Así mismo, la clave podrá ser una propiedad de este objeto. Por último, la búsqueda la podemos realizar a partir de propiedades del mismo objeto.

Clave: un valor a partir del cual los objetos almacenados son organizados. Por lo tanto, en cada contenedor no podemos repetir la clave. A continuación indicamos las características de la clave (llamada *key* en la especificación original de IndexedDB):

- **Un generador de claves.** Con el cual se producen nuevas claves de forma artificial y ordenada.
- **In-line Key.** La clave es almacenada como parte del valor.
- **out-of-line key.** La clave es almacenada fuera del valor almacenado.
- **key path.** Define el lugar desde donde el navegador extrae la clave en el contenedor.

Otro de los conceptos a tener en cuenta son los índices. Un índice es creado en un contenedor y permite buscar objetos dentro del mismo en base a los valores que contiene y no solamente en base a la clave del objeto.

La especificación de la API es extensa y actualmente todavía se encuentra como borrador de trabajo. Los conceptos del IndexedDB sobre los que trabajaremos a continuación serán los siguientes:

- Crear o abrir una base de datos e indicar su versión.
- Eliminar una base de datos.
- Iniciar una transacción.
- Realizar una petición de operación en base de datos.

Crear, abrir y eliminar una base de datos

Para la gestión de la base de datos es indispensable el método `open()`. Este método funciona de dos maneras: crea la base de datos si esta no existe o la abre en caso de que exista. Si todo funciona correctamente el evento `success` es disparado y la función indicada en la propiedad `onsuccess` es aplicada.

Como vemos en el código fuente, llamamos a la API de IndexedDB y aplicamos el método `open()` con el nombre de la base de datos. Junto con ello, asignamos a la propiedad `onsuccess` a la función “`respuesta_exitoo`”, la cual tiene un parámetro “`evento`”. El parámetro es un objeto que incluye información de la operación actualmente realizada. De hecho, la propiedad `evento.result` contiene el resultado de la operación, es decir la misma base de datos.

En caso existir un error se dispara la función relacionada con la propiedad `onerror` de la petición con el mismo evento de error como argumento. La API IndexedDB ha buscado minimizar el manejo de errores. Aun así, existen situaciones que producen errores, como por ejemplo cuando el usuario no permite al sitio web la creación de la base de datos.

```
var peticion = window.indexedDB.open("nombre");

peticion.onsuccess = respuesta_exitoo;
peticion.onerror = respuesta_error;

respuesta_exitoo(evento){
    var bbdd = evento.result;

    alert("Base de Datos abierta",bbdd);
};

respuesta_error(evento){
    alert(evento);
};
```

Uno de los atributos más importantes de la base de datos es la versión, que sirve como identificador único. La aplicación web podrá acceder en todo momento a una única versión de la base de datos. Este atributo es una cadena de caracteres inicializada al valor vacío una vez creada la base de datos.

A continuación detallamos cómo establecer la versión en la base de datos creada:

```
if(bbdd.version != "VERSION_1")
    var peticion = bbdd.setVersion("VERSION_1");

peticion.onsuccess = respuesta_exitoo;
peticion.onerror = respuesta_error;
```

Por último, la eliminación de una base de datos es similar a la creación. Una característica muy importante es que no se producirá ningún error en caso de intentar borrar una base de datos inexistente.

```
var peticion = window.indexedDB.deleteDatabase("nombre");

peticion.onsuccess = respuesta_exitoo;
peticion.onerror = respuesta_error;
```

Creación de contenedores e índices

En este ejemplo veremos cómo crear una estructura en la base de datos instanciada. Como primer paso, tenemos un objeto JavaScript estructurado en un *array* de dos elementos que representan contactos, con teléfono y dirección de correo electrónico.

Establecemos la versión de la base de datos, lo cual es similar a intentar abrir una conexión a la misma. En caso de que la versión de la base de datos no exista el evento será un error. En otro caso la petición finalizará con un evento exitoso. En el caso de que el evento sea exitoso se ejecutará la función incluida en la propiedad `onsuccess` de la petición.

En este caso la función está escrita de manera anónima y asignada directamente a la propiedad `onsuccess`. Por lo tanto, las operaciones asociadas con la alteración de la estructura de una base de datos la realizaremos en `setVersion()`, dentro de la ejecución de la función que es llamada en la propiedad `onsuccess`. En este caso, se realizan tres acciones.

La primera es la creación de un contenedor de objetos (similar a las tablas en el modelo de base de datos relacionales, ver Tabla 8.3). El objeto se denomina `contactos` y su `keyPath` será la variable `tel`. Es decir, que los objetos incluidos en el contenedor tendrán un valor de nombre `tel` que también será la clave.

Lo segundo es la creación de un índice mediante la función `createIndex()`. Se incluye el nombre del índice y el valor asociado al índice (en este caso será la variable `correo`). Por último, se incluye un *array* de parámetros opcionales que en este caso solo incluye la opción `unique`. Esto significa que podremos listar los objetos del *Object Store* por el valor de una variable denominada `correo` y que no pueden existir dos objetos con el mismo valor en la variable.

Lo tercero es la carga de objetos en el contenedor. En este caso simplemente se utiliza el método `add()` del contenedor y le pasamos como parámetro cada objeto JavaScript. Como vemos al inicio del ejemplo, los objetos son pares nombre/valor, siendo dos de los nombres `tel` y `correo`; el primero será la clave del contenedor y se construirá un índice para el segundo con la restricción adicional de que dos contactos no pueden tener el mismo correo.

```
contacto = [
    {tel: "1234", correo:"juan@mail.com", movil:"666123"},
    {tel: "2345", correo:"pedro@mail.com", movil:"664392"}
];

var peticion = bbdd.setVersion("VERSION_1");

peticion.onsuccess=function(event) {
    var contenedor = bbdd.createObjectStore("contactos",
        {keyPath:"tel"});
    contenedor.createIndex("correo", "correo", {unique:true});

    for (n in contacto){
        contenedor.add(contacto[n]);
    }
};
```

Operaciones con la base de datos

Las operaciones con las que se agregarán, modificarán y eliminarán datos debemos realizarlas en una transacción. En base de datos se indica como transacción a un conjunto de operaciones que se aplican formando una unidad de tal manera que dentro de la transacción se mantiene la integridad de los datos, haciendo que estas operaciones no puedan finalizar en un estado intermedio.

En caso de que se cancele la transacción en alguna operación intermedia, se empiezan a deshacer las modificaciones aplicadas hasta dejar la base de datos en su estado inicial, como si las operaciones dentro de la transacción nunca se hubiesen realizado. La transacción en IndexedDB está formada por tres parámetros:

El primer parámetro son los contenedores implicados en la transacción. En caso de no indicar nada se asume que todos los contenedores están en la transacción.

El segundo parámetro es el modo de la transacción que indica el tipo de operación a ser realizada y si dos transacciones pueden aplicarse de manera concurrente. La transacción puede ser abierta de tres modos diferentes:

- **Read_Only**. Solo lectura y sin modificación de datos (es la opción por defecto).
- **Read_Write**. Permite la lectura y escritura de datos en contenedores existentes.
- **Version_Change**. Permite cualquier tipo de operación, incluyendo la creación y borrado de contenedores e índices. Este tipo de transacción no se ejecutará de manera concurrente junto con ninguna otra transacción.

El tercer parámetro es la instancia de base de datos de la transacción. A continuación podemos ver un ejemplo de transacción donde la lista de contenedores asociados es solo uno. La transacción en este caso será de lectura/escritura; para indicarlo utilizamos el objeto enumerado `IDBTransaction` definido en la API de la especificación IndexedDB.

La transacción puede finalizar de tres maneras posibles: completada sin error, con error o abortada. Es necesario manejar estos tres eventos mediante diferentes funciones, como vemos en el ejemplo:

```
var transaccion = bbdd.transaction(["contactos"],  
    IDBTransaction.READ_WRITE);  
  
transaccion.oncomplete = function(evento){...};  
transaccion.onabort = función(evento){...};  
transaccion.onerror = function(evento){...};
```

Una vez que tenemos la transacción abierta podemos operar con los contenedores y objetos. Para agregar un nuevo elemento utilizamos la función `add()`. El resultado de agregar un nuevo objeto (`evento.result`) es la clave de dicho objeto añadido.

Por lo tanto, no es posible agregar dos objetos con la misma clave mediante la operación `add()`. En lugar de ello, para modificar un elemento existente utilizamos la función `put()`.

```
var contenedor = transaccion.objectStore("contactos");  
  
for (var i in contacto) {  
    var peticion = contenedor.add(contacto[i]);
```

```
peticion.onsuccess = function(evento) {...};  
}  
  
peticion = contenedor.put({tel:"2345",  
correo:"pedro@mail.com",movil:"661111"});
```

Por último, para eliminar un elemento utilizamos la función `delete()`. En el siguiente ejemplo llamamos al contenedor directamente a partir de la transacción y borramos el objeto cuya clave es "1234".

```
var peticion = bbdd.transaction(["contactos"],  
IDBTransaction.READ_WRITE).objectStore("contactos")  
.delete("1234");  
  
peticion.onsuccess = function(evento) {...};
```

La obtención de datos

La API permite obtener tanto objetos simples como conjunto de objetos. Para el caso de los objetos simples se utiliza el método `get()` con la clave del objeto a ser recuperado. El objeto devuelto se encontrará en la propiedad `result` del evento.

```
var transaccion = bbdd.transaccion(["contactos"]);  
var contenedor = transaccion.objectStore("contactos");  
var peticion = contenedor.get("2345");  
  
peticion.onerror = function(evento){...};  
  
peticion.onsuccess = function(evento){  
if(evento.result.tel=="2345")  
alert("Objeto recuperado correctamente!!!");  
};
```

En el caso de que busquemos obtener más de un objeto es necesario utilizar un cursor para recorrerlos, limitando o no el rango de objetos. En el ejemplo siguiente abrimos y recorremos un cursor a partir de un contenedor de objetos del tipo `contacto`. El cursor es el resultado del evento en caso de éxito (es decir, se encuentra en la propiedad `result`). La apertura del cursor se realiza mediante el método `openCursor()` pudiendo tener parámetros opcionales que limiten el rango.

En este caso el cursor no tiene ningún parámetro adicional por lo que recorrerá los contactos hasta finalizar. En la variable `cursor` se encuentra el objeto actual obtenido, de tal manera que con el método `continue()` cargamos la variable con el siguiente objeto (ver Figura 8.6).

```

var contenedor = bbdd.transaccion(["contactos"])
    .objectStore("contactos");

contenedor.openCursor().onsuccess = function(evento) {
    var cursor = evento.result;
    if (cursor) {
        alert("CONTACTO " + cursor.tel);
        cursor.continue();
    } else {
        alert("FINALIZADO");
    }
};

```

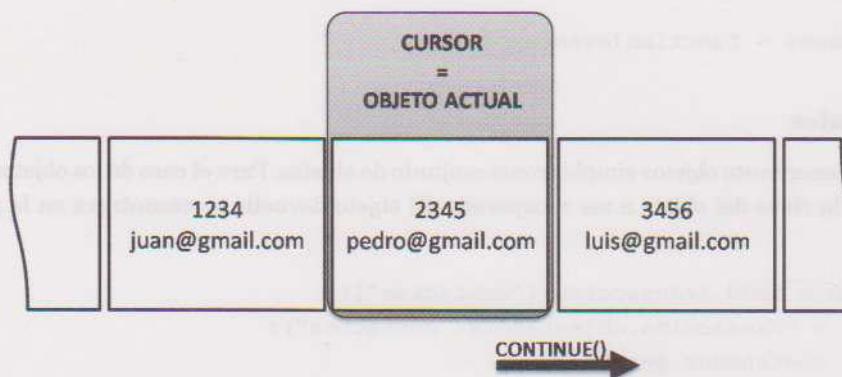


Figura 8.6. Esquema de un cursor en IndexedDB

Así mismo, existe la posibilidad de limitar el resultado obtenido por el cursor a un rango de claves (llamado *key range* en la API). Este rango de claves es flexible, pudiendo contener un único valor, o un único valor extremo (inferior o superior) o ambos valores extremos. A su vez las claves extremas pueden o no estar contenidas dentro del rango.

A continuación veremos algunos ejemplos. Suponiendo que tengamos en un contenedor llamado `contactos` un total de seis objetos cuya clave sea el teléfono de la siguiente manera: 1234, 2345, 3456, 4567, 5678 y 6789, el resultado de limitar el rango de acuerdo con lo indicado por la API de IndexedDB es:

- Solo un objeto cuya clave es 1234. Se utiliza la función `only()`:

```
rango = IDBKeyRange.only("1234");
```

- Cinco objetos, incluyendo a 2345: 2345, 3456, 4567, 5678 y 6789. Utilizamos la función `lowerBound()` con un parámetro indicando el límite inferior:

```
rango = IDBKeyRange.lowerBound("2345");
```

- Cuatro objetos, sin incluir a 2345: 3456, 4567, 5678 y 6789. Utilizamos la función `lowerBound()` con un parámetro adicional booleano indicando que no se excluye el límite (dato por el primer parámetro de la función), entre los objetos del resultado.

```
rango = IDBKeyRange.lowerBound("2345", true);
```

- Tres objetos, incluyendo a 3456: 1234, 2345 y 3456. Utilizamos la función `upperBound()`:

```
rango = IDBKeyRange.upperBound("3456");
```

- Tres objetos entre 2345 y 4567, sin incluir el límite inferior: 2345, 3456 y 4567. Utilizamos la función `bound()` que contiene como parámetros el límite inferior, el límite superior y el indicador de la exclusión de los valores límites.

```
rango = IDBKeyRange.bound("1234", "4567", true, false);
```

La variable `rango` es obtenida del elemento `IDBKeyRange` que forma parte de la API de IndexedDB y que debe ser implementado por el navegador. Esta variable se emplea como parámetro en el método `openCursor()` para la limitación de los objetos obtenidos.



¿SABÍAS QUE...?

Existen librerías que implementan una interfaz genérica para poder trabajar con IndexedDB o WebSQL de manera transparente.

ACTIVIDADES 8.2

- ▶ Investigue si es posible crear bases de datos en el cliente cuando realiza una navegación privada.
- ▶ Compare las diferentes implementaciones actuales de IndexedDB de los principales navegadores (Firefox, Internet Explorer y Chrome). ¿En qué se diferencian y qué similitudes tienen? ¿Es posible construir aplicaciones que hagan uso de IndexedDB de forma sencilla o serán necesarias librerías genéricas?

8.3 APLICACIONES EN CACHÉ

La caché de las aplicaciones web es el proceso de almacenar datos generados dinámicamente para que puedan ser utilizados nuevamente sin necesidad de realizar peticiones al servidor. Se utiliza principalmente para mejorar el rendimiento de las aplicaciones web y disminuir el tiempo de carga (Fielding et. al, 2008).

8.3.1 VENTAJAS Y DESVENTAJAS

Al igual que otras técnicas, esta también contiene un conjunto de ventajas y desventajas que la hacen beneficiosas solo en determinados ámbitos. Entre las ventajas más características encontramos principalmente dos:

- ✓ El rendimiento en las aplicaciones web es un aspecto fundamental. En ese sentido cualquier mejora provoca un cambio significativo en la experiencia del usuario.
- ✓ Es una tecnología ampliamente extendida y, por tanto, los navegadores web implementan componentes de caché utilizables por las aplicaciones.

Así mismo, entre las desventajas más reconocidas están las siguientes:

- ✓ En algunos escenarios (por ejemplo en aplicaciones ya construidas) puede ser complejo agregar aspectos de caché.
- ✓ El comportamiento de la aplicación puede verse afectado, especialmente si realizamos una mala programación.

8.3.2 ML 5

HTML 5 soporta actualmente la caché de los recursos de las aplicaciones web (W3C, 2008). El objetivo de estas nuevas características, junto con la API de IndexedDB es la posibilidad de obtener aplicaciones web y sitios web que funcionen correctamente cuando no tienen conexión con el servidor web (modo fuera de línea).

La funcionalidad consiste en obtener toda la información del sitio web antes de que sea explícitamente pedida por el usuario de tal manera que una vez almacenadas en la caché de aplicación puedan ser utilizadas en modo fuera de línea.

Como diferencia principal entre la caché habitual de una página web y la caché de aplicación es que en el segundo caso podremos acceder también a páginas que no hayan sido visitadas anteriormente.

En ese sentido el primer paso para realizar la caché de aplicación es la elaboración de un fichero llamado *MANIFEST* (o manifiesto en español). En este fichero listaremos el conjunto de recursos con los que se trabajarán en modo fuera de línea.

El manifiesto

Existen en total tres características que debemos tener en cuenta para utilizar la caché de aplicación de HTML 5 (en el caso de que esté implementado en el navegador). El primero de ellos es la realización del fichero manifiesto, el cuál será un fichero en texto plano con extensión “.appcache” y que contendrá los nombres de los ficheros y recursos en general que serán almacenados por el navegador para ser utilizados en modo fuera de línea.

El manifiesto comienza siempre con la cadena: “CACHE MANIFEST” y contendrá por lo menos tres partes, cada una con las URL absolutas o relativas al mismo manifiesto de los ficheros del sitio web. Las partes son las siguientes:

- **CACHE.** Las URL que se encuentran en esta sección serán mantenidas en la caché de la aplicación por el navegador para ser vistas en modo fuera de línea. Para cumplir con la sintaxis necesaria debe existir solo una URL por línea y debe apuntar a un único recurso (no se admiten caracteres comodín o anclas de página). De esta manera el navegador recibirá el manifiesto, lo leerá y mantendrá en la caché de aplicación todas las URL de esta sección.
- **NETWORK.** Las URL de esta sección no serán cargadas en la caché de aplicación. Es común utilizar el carácter comodín asterisco “*” para indicar que por defecto ninguna página será mantenida en la caché de aplicación (a no ser que se encuentre referenciada en la sección CACHE).
- **FALLBACK.** Indica el contenido que será mostrado en caso de que un recurso no sea encontrado, por ejemplo cuando la aplicación se encuentra fuera de línea y es necesario cargar un recurso que se encuentra en la sección NETWORK.

A continuación vamos a mostrar un ejemplo de manifiesto con las tres secciones. El nombre del fichero es “principal.appcache” y de acuerdo con lo descrito el fichero index.html estará en la caché de aplicación, junto con un fichero CSS y JavaScript. Así mismo, los ficheros “pago.html” y “ultimasnoticias.html” nunca estarán en la caché de aplicación. Por último, en caso de que mostremos las últimas noticias en el fichero “index.html”, en lugar de ello mostraremos el fichero “offline.html”.

CACHE MANIFEST

CACHE:

index.html
pagina.css
pagina.js

NETWORK:

pago.html
ultimasnoticias.html

FALLBACK:

offline.html

Referenciar el manifiesto

Una vez creado el manifiesto debemos referenciarlo dentro de la etiqueta HTML de las páginas HTML que estarán en la caché de aplicación. Por ejemplo, si analizamos el contenido de la página principal, esta hace referencia a otros dos recursos: la hoja de estilos CSS y el código de JavaScript. Los tres recursos están en el manifiesto y serán mantenidos en la caché.

```
<html>
  <head>
    <title>TITULO</title>
    <script src="pagina.js"></script>
    <link rel="stylesheet" href="pagina.css">
  </head>
  <body>
    <p>EN CONSTRUCCIÓN</p>
  </body>
</html>
```

Por lo tanto, lo que queda es incluir el atributo `manifest` con el nombre del fichero en la etiqueta HTML:

```
<html manifest="principal.appcache">
  ...
</html>
```

Mantener la integridad de la aplicación web

Una vez realizado ambos pasos descritos anteriormente el sitio web estará preparado para funcionar con la caché de aplicación. El funcionamiento interno recomendado por la W3C es el siguiente:

Al visitar el sitio web por primera vez se descargan los recursos que aparecen en el manifiesto. En caso de que sea la segunda vez que se visita el sitio web y si el contenido del manifiesto ha cambiado, se descargan nuevamente todos los recursos hacia la caché de aplicación.

El problema se encuentra cuando al pasar del modo fuera de línea al modo en línea se ha actualizado en el servidor algún recurso que se encuentre en la caché de aplicación. Debido a que el manifiesto no se ha modificado (porque el recurso no ha cambiado de nombre), tampoco se recarga la página desde el servidor. Como resultado de esto se sigue mostrando la versión antigua contenida en la caché de aplicación.

La solución es actualizar el manifiesto. Y para no modificar el contenido del fichero recomendamos incluir un comentario con la versión o fecha de última actualización del manifiesto, para ir modificándolo a partir de los cambios del contenido de los recursos listados.



¿SABÍAS QUE...?

Los navegadores actualmente implementan mecanismos para conocer el estado de la caché de aplicación.

ACTIVIDADES 8.3



- Investigue los navegadores que actualmente implementan la caché de aplicación y sus características especiales.
- ¿De qué manera puedo conocer, en mi aplicación del lado del cliente si el navegador se encuentra en línea o fuera de línea? Sugerencia: revisar la especificación *Offline Web Applications* de la W3C: www.w3.org/TR/offline-Webapps/



RESUMEN DEL CAPÍTULO

En este capítulo hemos presentado los conceptos y tecnologías principales del almacenamiento de datos del lado del cliente, así como la posibilidad del trabajo fuera de línea de HTML 5. Hemos estudiado las características de los objetos DOM *Storage* y su relación con la tecnología anterior más utilizadas: las *cookies*.

HTML 5 soporta actualmente la caché de los recursos de las aplicaciones web. El objetivo de estas nuevas características, junto con la API de IndexedDB es la posibilidad de obtener aplicaciones web y sitios web que funcionen correctamente cuando no tienen la conexión con el servidor web (modo fuera de línea).

IndexedDB es una API para el almacenamiento de gran cantidad de datos de manera estructurada y es el reemplazante natural de WebSQL cuya especificación ha sido abonada en el año 2010 por la W3C. De esta manera pasamos de una base de datos relacional a una orientada a objetos JavaScript para realizar una persistencia directa a partir de los contenedores y utilizando índices y cursores para las consultas.

Por último, en HTML 5 aparece el concepto de caché de aplicación. La funcionalidad consiste en obtener toda la información del sitio web antes de que sea explícitamente pedida por el usuario de tal manera que una vez almacenadas en la caché de aplicación podrán ser utilizadas en modo fuera de línea.