MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Almacenamiento web

14

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Fundamentos del almacenamiento 4

/ 3. Tipos de almacenamiento en cliente 4

/ 4. Objetos, propiedades y métodos del almacenamiento web 5

/ 5. Caso práctico 1: “Robo de datos” 5

/ 6. Almacenamiento local y a nivel de sesión 6

/ 7. Eventos de almacenamiento 6

/ 8. Ejemplo de almacenamiento web 7

/ 9. Bases de datos en entorno cliente 8

/ 10. Aplicaciones en caché 9

/ 11. Caso práctico 2: “Caché, ¿sí o no?” 9

/ 12. Resumen y resolución del caso práctico de la unidad 10

/ 13. Bibliografía 10

OBJETIVOS

Comprender el funcionamiento del almacenamiento web.

Saber identificar diferentes tipos de almacenamiento web.

Conocer la base de datos en cliente.

Saber introducir la vigencia de los valores almacenados.

Conocer las restricciones del almacenamiento web.

Identificar riesgos en el almacenamiento web.

Saber utilizar el almacenamiento permanente y temporal.

Conocer las aplicaciones web basadas en caché.

/ 1. Introducción y contextualización práctica

La tecnología web está en constante evolución, ya que dos de sus pilares fundamentales como HTML y JavaScript aumentan constantemente sus capacidades.

Por ejemplo, en el caso de JavaScript, lo que comenzó como un lenguaje para realizar determinadas tareas para facilitar la interacción con el usuario e implementar medidas de control, se convirtió en un lenguaje que puede ser utilizado tanto en el cliente como en el servidor.

Por ello, no es de extrañar que JavaScript y HTML ofrezcan un gran número de posibilidades como, por ejemplo, realizar almacenamiento en el cliente con diferentes estrategias.

Gracias a la evolución de estas tecnologías, otros ámbitos de desarrollo, como el móvil, se han visto beneficiados. En la actualidad, es posible crear aplicaciones móviles con lenguajes como JavaScript y HTML5.

Escucha el siguiente audio que describe el caso práctico que iremos resolviendo a lo largo de la unidad.

Fig. 1. Almacenamiento web.

Audio Intro. "Almacenamiento con JavaScript"

http://bit.ly/2WmzGnw

/ 2. Fundamentos del almacenamiento

Una aplicación web puede utilizar las funcionalidades del navegador para almacenar datos en la máquina del usuario. **El almacenamiento en el lado del cliente requiere que el navegador pueda tener espacio de memoria asignado para dicho almacenamiento.**

Por ejemplo, es posible que una aplicación web pueda almacenar las preferencias del usuario o incluso pueda guardar datos sobre el estado de ejecución, con el fin de que si sucede un error, poder continuar el flujo de ejecución justo en el punto por el que iba el usuario en su última visita.

Por defecto, **el almacenamiento en el cliente está dividido por página**, lo que significa que una página no puede acceder a los datos de otra página. Sin embargo, si ambas páginas provienen del mismo dominio, podrán compartir datos.

Las aplicaciones web pueden **fijar el tiempo de vida que los datos pueden permanecer almacenados**, es decir, se puede optar por un almacenamiento temporal o permanente:

- En el almacenamiento temporal, los datos se almacenan únicamente mientras la ventana del navegador exista.
- En el almacenamiento de manera permanente, los datos persisten durante un tiempo indeterminado, aunque el navegador se cierre.

Almacenamiento y seguridad

A la hora de realizar el almacenamiento en el cliente, es importante seguir algunas medidas de seguridad de los datos que se están almacenando, especialmente si van a permanecer almacenados de manera indefinida de forma local en el cliente.

En la actualidad, muchos navegadores ofrecen la posibilidad de guardar las contraseñas de los usuarios. Para ello, utilizan diferentes mecanismos de encriptación. Nótese que los ejemplos que mostramos a lo largo de este tema no se centran en encriptar los datos, por lo que será responsabilidad del programador web realizar dicha encriptación.

Video 1. "Seguridad en el almacenamiento con JavaScript"
https://bit.ly/3gVyAZo

/ 3. Tipos de almacenamiento en cliente

Existen diferentes formas de poder realizar el almacenamiento en el lado del cliente:

- **Almacenamiento web.** Este tipo de almacenamiento consiste en la utilización del API de JavaScript que permite el almacenamiento local o a nivel de sesión. A través de esta funcionalidad, es posible almacenar datos en el formato de pares clave-valor. En teoría, la utilización del almacenamiento web es relativamente sencilla y se puede adaptar para almacenar una cantidad de datos considerable.
- ***Cookies.*** Se trata de un almacenamiento antiguo en el lado del cliente que fue diseñado para permitir al servidor guardar cierta cantidad de información en el cliente. Un problema de las *cookies* es que son legibles mediante *scripts* en el cliente y solo pueden almacenar pequeñas cantidades de datos. Además, las *cookies* son transmitidas al servidor en cada petición HTTP, aunque esos datos no se utilicen para el procesamiento de la petición realizada.

Fig. 2. El navegador puede ser un almacén de datos.

- **IndexedDB.** Se trata de un API para el acceso al objeto de base de datos que permite indexación.

Audio 1. "Bases de datos"
http://bit.ly/38cQaDY

/ 4. Objetos, propiedades y métodos del almacenamiento web

El objeto *window* ofrece dos propiedades para poder realizar el almacenamiento web: *localStorage* y *sessionStorage*.

Un objeto de almacenamiento JavaScript se comporta de manera muy similar al resto de objetos del lenguaje, exceptuando lo siguiente:

- Los valores almacenados tienen que ser cadena de caracteres.
- Las propiedades almacenadas en el objeto de almacenamiento son persistentes. Esto quiere decir que si guardamos determinadas propiedades en el objeto y el usuario vuelve recargar la página, las propiedades seguirán existiendo.

Al igual que sucede con otros objetos, es posible borrar una propiedad utilizando el operador ***delete()***. A su vez, también es posible mostrar todo el contenido almacenado utilizando cualquier sentencia iterativa. Es posible borrar todo el contenido almacenado utilizando el método ***clear()***.

El API de almacenamiento define los siguientes métodos:

- ***getItem()***: permite obtener un determinado elemento.
- ***setItem()***: permite guardar un elemento con su clave asociada.
- ***removeItem():*** permite borrar un elemento a partir de su clave.


Fig. 3. Almacenamiento en la web.

Es importante recordar que **los objetos de almacenamiento solo permiten guardar contenido en formato cadena**. En el caso que se necesite guardar otro tipo de datos, hay que realizar una conversión de estos manualmente.

/ 5. Caso práctico 1: "Robo de datos"

Planteamiento: Una aplicación web que teníamos en producción ha sufrido un ataque y han robado datos que nuestros usuarios tenían almacenados en su navegador.

Nuestro equipo de seguridad nos traslada que aunque no había ningún mecanismo de encriptación, nos quedemos tranquilos, ya que ningún tercero podría acceder a dichos datos.

Nudo: ¿Crees que existe algún riesgo real?

Desenlace: Como hemos visto en el apartado de almacenamiento y seguridad, es muy importante considerar que los datos que se guardan en el cliente no están encriptados y, por lo tanto, son visibles por todo el que quiera leerlos.

Nuestro equipo de seguridad no podría estar más equivocado, pues al producirse el robo de datos, estos son legibles por terceras personas (programas, aplicaciones o robots) que pueden tener objetivos fraudulentos.

Para poder solucionar esto, es importante que se minimicen los datos que almacenamos en el cliente, y al almacenarlos, utilizar criptografía para ello y que sean datos no sensibles.

Fig. 4. La seguridad informática debe estar presente en todos los bloques de nuestros desarrollos.

/ 6. Almacenamiento local y a nivel de sesión

a. Almacenamiento local

A través de ***localStorage***, los datos se almacenan de manera permanente, es decir, **no caducan y se almacenan en el dispositivo del usuario hasta que una aplicación web los borre o el usuario realice el borrado** a través del propio navegador web.

Este almacenamiento está ligado al origen del documento web, teniendo en cuenta el protocolo, el nombre del servidor y el puerto de conexión. Así, todos los documentos con el mismo origen pueden compartir los datos de almacenamiento local. También hay que considerar que estos datos van a depender del navegador utilizado, es decir, aunque se cumpla lo anterior, si el navegador cambia los datos, estos dejarán de estar disponibles.

b. Almacenamiento a nivel de sesión

El almacenamiento a nivel de sesión tiene un **tiempo de vida** diferente al almacenamiento local, **que dependerá del tiempo que la ventana del navegador** (o la pestaña correspondiente) **permanezca abierta**.

Cuando se cierran alguna de las dos, los datos almacenados desaparecerán.

Hay que tener en cuenta que **los navegadores actuales son capaces de volver a reabrir las pestañas de navegación y restaurar los datos de sesión**. En este contexto, el tiempo de vida de los datos de sesión es más largo de lo que el programador podría esperar.

Como sucede con el almacenamiento local, el almacenamiento de sesión también está **segregado por el origen**, es decir, solo se pueden compartir con aquellos sitios que provengan del mismo dominio.

Además, en el almacenamiento a nivel de sesión, **los datos también están divididos por pestañas o ventanas**. Esto quiere decir que diferentes pestañas o ventanas no podrán compartir los datos de sesión que se guarden en el cliente.

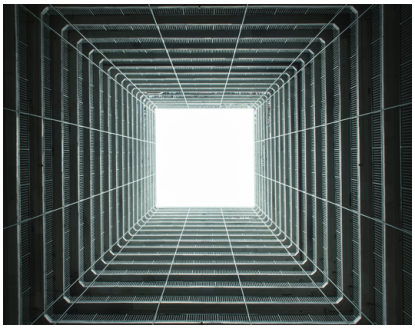

Fig. 5. JavaScript permite el almacenamiento a nivel de sesión.

/ 7. Eventos de almacenamiento

Independientemente del tipo de almacenamiento utilizado, **los navegadores pueden lanzar un evento de almacenamiento en aquellas ventanas donde los datos pueden ser visibles.** Sin embargo, este evento no se lanza en la ventana que ha realizado el almacenamiento. **Es el navegador el que determina qué ventanas tienen acceso a qué datos.**

Para lograr este comportamiento, es necesario registrar un evento de almacenamiento en la ventana cuando se carga el *script*. El evento asociado almacenamiento es ***window.onstorage()***.

Al lanzarse este evento, hay que tener en cuenta algunas propiedades importantes:

- **key:** hace referencia al identificador del elemento que se ha insertado o borrado del almacenamiento. En el caso de borrar todo el almacén de datos invocando a la función **clear()**, la clave será ***null***.
- **newValue:** contendrá el nuevo valor del elemento a almacenar. En el caso de un evento de borrado, este campo estará vacío.
- **oldValue:** contendrá el valor previo en el almacén de datos. Si se trata de una nueva clave, esta propiedad estará vacía.
- **storageArea:** hace referencia al objeto utilizado para el almacenamiento.
- **url:** se refiere al documento cuyo *script* ha realizado el cambio de almacenamiento.

Fig. 6. Los eventos de almacenamiento se lanzan en la ventana.

/ 8. Ejemplo de almacenamiento web

Antes de utilizar cualquier API de almacenamiento, es necesario **comprobar si el navegador que estamos utilizando lo soporta**.

En el siguiente código, se realiza la comprobación correspondiente para saber si nuestro navegador es capaz de almacenar datos.

```
if (typeof(Storage) !== "undefined") {
  //El navegador permite almacenamiento

} else {
  //El navegador no soporta el almacenamiento

}
```

Código 1. Comprobación si se permite almacenamiento en navegador.

En el siguiente ejemplo, se realiza un almacenamiento local permanente y, posteriormente, se leen los datos para visualizarlos en el documento HTML. Por último, se borra el valor de la clave utilizada.

```
// Almacenamiento local (permanente)
localStorage.setItem("surname", "Gómez");

// Recuperación de los datos
document.getElementById("info").innerHTML = localStorage.getItem("surname");
localStorage.removeItem("surname ");
```

Código 2. Almacenamiento permanente y borrado.

En el siguiente ejemplo, se muestra un almacenamiento a nivel de sesión.

En este caso, el código comprueba si existe el identificador ***numClicks*** declarado previamente. En caso de existir, se incrementa en 1; en caso contrario, se inicializa 1.

Por último, se actualiza el documento web con el número de clics realizados durante la sesión. En este ejemplo no es necesario borrar los datos, pues al cerrar el navegador desaparecen.

```
if (sessionStorage.numClicks) {
 sessionStorage.numClicks = Number(sessionStorage.numClicks) + 1;
} else {
 sessionStorage.numClicks = 1;
}
document.getElementById("result").innerHTML = "Haz pulsado el botón " +
sessionStorage.numClicks + " veces en esta sesión.";
```

Código 3. numClicks.

/ 9. Bases de datos en entorno cliente

La arquitectura tradicional de una aplicación web se centra tradicionalmente en las características de HTML, CSS y JavaScript en el cliente, y en el procesamiento de peticiones y acceso a datos en el servidor. Sin embargo, **JavaScript ofrece la posibilidad de utilizar una base de datos en el cliente para almacenar datos de manera persistente.**

El objeto ***IndexedDB*** permite almacenar datos utilizando una base de datos no relacional en el cliente. Una **ventaja** con respecto al almacenamiento local/sesión (basado en clave-valor) es que permite realizar un almacenamiento más potente, eficaz y robusto.

El funcionamiento de esta base de datos es relativamente sencillo. En primer lugar, hay que abrir una base de datos indicando su nombre.

A continuación, se crea un objeto transacción con la tarea asociada. Por último, se ejecuta la transacción sobre la base de datos.

Para lograr este funcionamiento, la base de datos debe existir previamente.

```
function initdb(db, callback) {
  let store = db.createObjectStore("zipcodes", // store name
                    { keyPath: "zipcode" });
  store.createIndex("cities", "city");
  fetch("zipcodes.json")
    .then(response => response.json())
    .then(zipcodes => {
      let transaction = db.transaction(["zipcodes"], "readwrite");
      transaction.onerror = console.error;
      let store = transaction.objectStore("zipcodes");
        for(let record of zipcodes) { store.put(record); }
      transaction.oncomplete = () => { callback(db); };
    });
}
```

Código 4. Funcionamiento de la base de datos en cliente.

Vídeo 2. "Bases de datos SQL y NoSQL"
https://bit.ly/2WkimPP

/ 10. Aplicaciones en caché

El funcionamiento de una aplicación web está **fundamentado en tener una conexión permanente con la red.** En el caso de no disponer de conexión, la aplicación no funcionará. A través de **HTML5**, es posible **utilizar un mecanismo para ejecutar aplicaciones web sin conexión permanente**.

Como sabemos, una aplicación web está formada por diferentes páginas en las que podamos navegar y que realizarán diferentes peticiones al servidor de determinados recursos. Estos recursos pueden ser documentos HTML, CSS, JavaScript, imágenes o cualquier otro elemento necesario para poder crear una visualización de la web.

Para lograr un **funcionamiento sin conexión**, es necesario indicar al navegador que debe descargar el contenido cuando conecta con diferentes páginas. Esto se realiza a través del archivo ***manifest*** que HTML5 permite definir. De esta forma, cuando el navegador conecta con una página, lo primero que hace es buscar este archivo y descargar el contenido indicado en él.

Es responsabilidad del desarrollador web crear el archivo *manifest* para permitir el funcionamiento sin conexión. Concretamente, en un archivo de texto que formará parte del documento web y que estará disponible también en el servidor.

```
<!—- index.html -->
<!DOCTYPE HTML>
<html manifest="cache.appcache">
<head>
 <title>MyAppOffline</title>
 <script src="test.js"></script>
 <link rel="stylesheet" href="styles.css">
</head>
<body>
 Prueba del archivo manifiesto
</body>
</html>

//Archivo cache

CACHE MANIFEST
/styles.css
/test.js
/test.png
```

Código 5. Aplicaciones en caché.

/ 11. Caso práctico 2: "Caché, ¿sí o no?"

Planteamiento: Nuestro equipo de desarrollo decide utilizar al máximo las funcionalidades que nos ofrece HTML5 para poder generar aplicaciones que puedan utilizar siempre los datos en caché y solo conectarse al servidor una vez al día. De esta forma, nos comentan que los usuarios pueden ahorrar en consumo de datos.

Nudo: ¿Qué opinas de esta solución?

Desenlace: Como hemos visto en el apartado sobre la utilización de la caché, es posible que una aplicación realice el almacenamiento temporal de datos para poder utilizar la web cuando no disponga de conexión con la red.

Al utilizar este enfoque de manera sistemática, podemos tener un problema de actualización de datos. Si nuestra aplicación necesita datos en tiempo real, el enfoque con las cachés no es una buena solución.

Otro punto que tener en cuenta es que cada vez que se conecta la aplicación web, deberá descargar todos los datos, pudiendo consumir más datos que el enfoque sin cachés, en el que los usuarios solo consumen lo que visualizan.

Fig. 7. Caché, ¿sí o no?

/ 12. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos presentado las diferentes capacidades que tiene JavaScript para realizar el **almacenamiento en el lado del cliente**. Hemos podido observar que el **API de JavaScript** permite realizar un almacenamiento temporal o permanente atendiendo a diferentes enfoques.

A lo largo de este tema, hemos revisado el API de JavaScript para el **almacenamiento permanente** utilizando el objeto para almacenamiento local. También hemos revisado el API para el **almacenamiento a nivel de sesión** en el navegador web. Una de las funcionalidades que puede resultar más extraña es la utilización de bases de datos en el cliente, pues tradicionalmente se ha utilizado siempre en el servidor. Es importante destacar que estas bases de datos nunca sustituirán a las bases de datos del servidor.

En relación con el almacenamiento, es fundamental y responsabilidad del desarrollador web **securizar correctamente los datos almacenados** con el fin de evitar posible robo de datos.

Resolución del caso práctico inicial

Como hemos visto en este tema, JavaScript permite realizar el almacenamiento de datos en el navegador del cliente. A través del API de JavaScript, podemos almacenar datos en diferentes estructuras internas del navegador.

Como se nos dice, la decisión del equipo es realizar un almacenamiento permanente en el sistema de archivos. Sin embargo, esto no es posible, ya que cuando se ejecuta JavaScript desde el entorno de un navegador, no se puede acceder al sistema de archivos.

A esto hay que sumar el hecho de que no es una buena decisión trasladar esta responsabilidad al cliente por motivos de seguridad.

Por ejemplo, puede existir un robo de estos datos y el usuario no sabe quién está utilizando sus credenciales.

Fig. 8. Para tomar una decisión, es necesario evaluar diversos aspectos.

/ 13. Bibliografía

Frisbie, M. (2020). P*rofessional JavaScript® for web developers.* Indianapolis, Indiana: Wrox, a Wiley brand.

Camden, R. (2016). *Client-side data storage: keeping it local.* Sebastopol, CA: O'Reilly Media.