Programación

Almacenamiento

Decimonovena semana

Marzo 2022

Cruz García, Iago

Introducción

Vamos a trabajar los conceptos de almacenamiento de información en el lado del cliente, del usuario. Esto nos permitirá mantener datos de una sesión a otra, como datos de guardado en una partida u opciones como el idioma o el formato de la página.

Recomendación: A partir de ahora es necesario que a la hora de realizar ejercicios nos acostumbremos a buscar información en la documentación oficial. En caso de no poder resolver las dudas con la documentación ofrecida, el siguiente paso será preguntar las dudas en el foro de clase.

Documentación oficial de JavaScript

Foro de la asignatura

Documentación sobre almacenamiento en web

Documentación sobre cookies

Almacenamiento

En el desarrollo de aplicaciones web, tenemos múltiples formas de almacenar información de nuestras aplicaciones, tanto datos útiles para los desarrolladores como métricas de uso o información de los datos que se necesiten como almacenamiento que facilita al usuario el uso de la web como son propiedades de la página, opciones de personalización o datos de guardado.

Para realizar esto, utilizaremos la API que proporcionan los navegadores a través del objeto 'window' conocida como 'Web Storage API', una serie de herramientas que facilitan el uso y almacenamiento de información en el navegador, evitando utilizar la documentación de 'cookies' que es algo obtusa.

Vamos a utilizar y notar las diferencias entre los dos mecanismos que ofrece esta API, LocalStorage y SessionStorage.

SessionStorage

Solo almacena la información mientras el usuario mantenga abierta la pestaña o el navegador.

index.html

```
<html id="html">

<head></head>

<body id="cuerpo">
	<button id="boton1">Contador++</button>
	<button id="boton2">Guardar contador</button>
	<button id="boton3">Mostrar contador</button>
	<button id="boton4">Eliminar contador</button>
	<button id="boton5">Eliminar todo</button>
</body>
<footer>
	<script src="main.js"></script>
</footer>

</html>
```

main.js

```
function __main__() {
	var contador = 0;
	var boton1 = document.getElementById("boton1");
	var boton2 = document.getElementById("boton2");
	boton1.addEventListener("click", (event) => {
	contador++;
	});
	boton2.addEventListener("click", (event) => {
	sessionStorage.setItem("contador", contador);
	});
	boton3.addEventListener("click", (event) => {
	var dato = sessionStorage.getItem("contador");
	console.log(dato);
	});
	boton4.addEventListener("click", (event) => {
	sessionStorage.removeItem("contador");
	});
	boton5.addEventListener("click", (event) => {
```

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

__main__();
```

Como vemos:

- **setItem(clave,valor):** Permite guardar el **valor**, en este caso, lo que contenga la variable *contador* en el almacenamiento de sesión, bajo la **clave** "contador". Esta clave podría ser un nombre cualquiera, no tiene porqué coincidir con el nombre de la variable del dato.
- **getItem(clave):** Recoge el **valor** del dato almacenado en la sesión bajo la **clave** dada, en este caso "contador".
- **removeItem(clave):** Elimina el **valor** almacenado dada una clave.
- **clear():** Elimina todos los valores almacenados en la sesión.

A tener en cuenta, todos los valores almacenados se convierten a cadenas de caracteres o *string*, por lo que cualquier valor numérico almacenado debe de tratarse con cuidado si se desean realizar operaciones a posteriori.

LocalStorage

A diferencia de SessionStorage permite el almacenamiento entre sesiones, cerrar la pestaña, cerrar el navegador o recargar la página no afecta a los datos guardados.

Los métodos y su estructura son la misma, simplemente cambia la sintaxis, por lo que os dejo a continuación el *main.js* para que probeis ambas implementaciones por separado y ver las diferencias.

main.js

```
function __main__() {
        var contador = 0;
        var boton1 = document.getElementById("boton1");
        var boton2 = document.getElementById("boton2");
        boton1.addEventListener("click", (event) => {
        contador++;
        });
        boton2.addEventListener("click", (event) => {
        localStorage.setItem("contador", contador);
        });
        boton3.addEventListener("click", (event) => {
        var dato = localStorage.getItem("contador");
        console.log(dato);
        });
        boton4.addEventListener("click", (event) => {
        localStorage.removeItem("contador");
        });
        boton5.addEventListener("click", (event) => {
        localStorage.clear();
        });
}

__main__();
```

Otros métodos

A continuación se presentan otras metodologías para el almacenamiento de información. Son más arcaicos, poco frecuentes o incluso poco aconsejables, pero tener la información sobre ellos os puede ayudar en el futuro, por lo que os muestro unos ejemplos.

Cookies

Utilizadas todavía a día de hoy, son capaces de almacenar datos igual que los dos métodos principales, con la salvedad de que permite personalizar de forma más amplia las opciones de guardado y de como se tratan esos datos, permitiendo escoger el tiempo de almacenamiento hasta su borrado automático o si se permite almacenar entre las distintas páginas que componen una aplicación web.

MDN desaconseja su uso si no se tratan con precaución, pues puede conllevar un uso excesivo de memoria o distribución de datos sin permiso.

main.js

```
function __main__() {
        var contador = 0;
        var boton1 = document.getElementById("boton1");
        var boton2 = document.getElementById("boton2");
        var boton3 = document.getElementById("boton3");
        boton1.addEventListener("click", (event) => {
        contador++;
        });
        boton2.addEventListener("click", (event) => {
```

```
        document.cookie = "contador=" + contador + ";SameSite=lax;";
        document.cookie = "prueba1=probando;SameSite=lax;";
        });
        boton3.addEventListener("click", (event) => {
        let cook = document.cookie;
        alert(cook);
        });

__main__();
```

Nótese la sintaxis utilizada, donde:

- **Document.cookie = '"clave=valor;"':** Realiza la misma función que **setItem(clave,valor)**. Debe escribirse el ';' punto y coma al final para indicar cuando termina el valor.
- **SameSite=opción;:** Una de las variadas opciones que admite el navegador para las cookies. Podéis ver todas las opciones en el enlace de MDN.
- **Document.cookie:** Asignándolo a una variable, nos permite recoger todos los datos que se almacenan. Para encontrar el que buscamos, utilizaremos la sintaxis que nos brinda MDN:

```
const cookieValue = document.cookie
  .split('; ')
  .find(row => row.startsWith('clave='))
  .split('=')[1];
```

Esto separará los valores por filas cuando encuentre ';' el punto y

coma y buscará la fila que comience por la clave y recoge el valor en la segunda posición [1], que será el **valor** de nuestra cookie.

Ficheros

Si bien son muy utilizados en otros lenguajes dada la capacidad de reusabilidad y facilidad de lectura y multiplataforma, JavaScript no permite estas funciones. Esto es debido a la baja seguridad que esto proporciona a través de la web.

Supongamos que pudieramos crear o acceder a ficheros que tienen los usuarios. En manos inocentes, esto nos permitiría almacenar datos y tratarlos de forma rápida y sencilla, además de permitir a un usuario trasladar opciones, partidas, etc. de una máquina a otra.

El problema surge cuando esa inocencia desaparece, pues tener acceso a un solo fichero de una máquina implica acceso total a dicha máquina, con acceso de administrador o super usuario, con las consecuencias que eso conlleva: desde lectura de datos comprometidos hasta los ataques cibernéticos.

Bases de datos

Veremos a lo largo del ciclo que podemos hacer uso de bases de datos de dos maneras: desde el propio cliente o desde el servidor.

Como ambas formas son relativamente complejas y avanzadas, este apartado es meramente informativo acerca de su posible uso.

Esta forma de almacenamiento nos permite expandir en las opciones y el trato de datos de los usuarios, donde podríamos manejar la capacidad de crear cuentas de usuario o consulta masiva de datos.