UT08.2: Almacenamiento

Contenido

1. Introducción

En este documento vamos a trabajar el almacenamiento de información en el navegador, para poderla recuperar posteriormente, independientemente de que se recargue la página. Tenemos básicamente dos mecanismos para almacenar información en formato nombre-valor.

- Cookies.
- API Web Storage.

2. Cookies

Como ya hemos estudiado en la primera unidad del curso, una cookie es un par nombre y valor que almacenamos en el navegador en un archivo de texto. Las cookies son parte del protocolo HTTP.

La principal función de las cookies es mantener el identificador de la sesión en una aplicación web, lo que nos permite identificar los clientes en el lado del servidor manteniendo su estado, ya que el protocolo HTTP es un protocolo sin estado.

A modo de resumen los atributos de una cookie son:

- Par nombre-valor: Almacena un dato en formato string asociándole un nombre para su recuperación. Este atributo es obligatorio.
- Expiración: Fecha en la que expirará la cookie. Pasada esa fecha no podremos acceder a su valor ya que el navegador la destruirá.
- Path: Es la ruta dentro de nuestra web a la que estaremos asociando la cookie. Fuera de esa ruta, la cookie es inaccesible.
- Dominio: Es el dominio a la que hemos asociado la cookie. Las cookies siguen el protocolo de seguridad del mismo origen, por lo que no podremos leer ni escribir cookies de otro dominio o puerto al que estamos utilizando.
- Seguridad: Si se aplica la seguridad, la cookie solo será accesible mediante protocolo HTTPS.

Las cookies pueden ser tratadas desde el cliente como desde el servidor.

2.1. Gestión de cookies

El acceso se hace a través del *API DOM* a través del objeto `document.cookie`. Por lo que crear una cookie es tan fácil como asignarlo en el objeto. En el siguiente ejemplo vemos como crear dos cookies asociadas al dominio sobre el que estamos accediendo la página, así como el path. Las cookies deben crearse de una en una.

```
document.cookie = 'Cookie1 = valor1';
document.cookie = 'Cookie2 = valor2';
```

Accediendo al objeto podremos obtener las cookies almacenadas en el dominio y en el path en el que estamos ubicados concatenadas en un *string*. El resultado de las cookies anteriores sería el siguiente:

```
Cookie1=valor1; Cookie2=valor2
```

Para modificar el valor de una cookie basta asignarle un nuevo valor al nombre de la cookie.

```
document.cookie = 'Cookie1 = Nuevo valor1';
document.cookie = 'Cookie2 = Nuevo valor2';
```

Trabajar con las cookies se hace tedioso porque no tenemos un API específica, por lo que tenemos que parsear la cadena para obtener un valor concreto. El siguiente código obtiene el valor de la cookie 'Cookie1' utilizando una expresión regular.

```
let cookie1 =
document.cookie.replace(/(?:(?:^|.*;\s*)Cookie1\s*\=\s*([^;]*).*$)|^.*$/,
"$1");
```

2.2. Atributos opcionales

Podemos jugar con los atributos opcionales de la cookie.

- `;path=path`. Si no se especifica, por defecto corresponde a la ruta del documento actual. Lo habitual es utiliza la ruta '/' que indica la raíz de la web, por lo que la cookie estará accesible en toda la aplicación.
- `;domain=domain`. Si no se especifica, su valor por defecto es la dirección de la página actual. Si se especifica un dominio, los subdominios siempre son incluidos. No es habitual utilizarlo.
- `;max-age=31536000`. Duración máxima de la cookie expresada en segundos. En este ejemplo le damos un año.
- `;expires=fecha-en-formato-GMTString`. Si no se especifica `max-age` ni `expires`, la cookie expirará al terminar la sesión actual.
- `;secure` La cookie sólo será transmitida en un protocolo seguro.
- `;samesite` Este atributo impide al navegador enviar esta cookie a través de peticiones cross-site. Los valores posibles son `lax` o `strict`.
    - `strict`: impide que la cookie sea enviada por el navegador al sitio destino en contexto de navegador cross-site, incluso cuando sigue un enlace regular.
    - `lax`: sólo envía cookies a las peticiones de GET. Es suficiente para seguir al usuario, pero evitará muchos ataques CSRF.

El siguiente código crea una cookie con una duración máxima de 60 segundos.

```
document.cookie = 'expiredCookie = ' +
      new Date().toLocaleTimeString() + ';max-age=60;path=/';
```

Para recuperarla debemos recargar la página. Durante los primeros 60 segundos estará disponible, pasado ese tiempo al recargar la página la cookie habrá caducado y no estará accesible.

```
let expiredCookie =
document.cookie.replace(/(?:(?:^|.*;\s*)expiredCookie\s*\=\s*([^;]*).*$)|
^.*$/, "$1");
```

2.3. Funciones útiles

En la página de la web W3Schools (https://www.w3schools.com/js/js_cookies.asp) podemos encontrar algunas funciones para reutilizar en nuestro código.

La función `setCookie()` recibe tres argumentos, el nombre de la cookie, el valor asignado, y el tiempo de expiración expresado en días. La cookie es creada directamente en la raíz de la aplicación.

```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires="+ d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

La función `getCookie()` permite recuperar una cookie a partir de su nombre.

```
function getCookie(cname) {
  let re = new RegExp('(?:(?:^|.*;\\s*)' + cname +
'\\s*\\=\\s*([^;]*).*$)|^.*$');
  return document.cookie.replace(re, "$1");
}
```

Veamos un ejemplo. La siguiente función muestra el nombre del usuario en una alerta en función de la cookie *username* o solicita el nombre del usuario para almacenarla.

```
function greetUser() {
  let user = getCookie("username");
  if (user) {
    alert("Hola " + user);
  } else {
    user = prompt("Dime tu nombre:", "");
    if (user != "" && user != null) {
      setCookie("username", user, 10);
    }
  }
}
```

2.4. Borrado de cookies

Para borrar una cookie basta con asígnale una fecha pasada, siendo su valor irrelevante.

```
setCookie("username", '', 0);
```

2.5. Problemas con las cookies

Básicamente tenemos dos problemas con las cookies.

- El tamaño para almacenar cookies es limitado en el navegador.
- Las cookies viajan en cada petición según la especificación del protocolo HTTP, con lo que si guardamos mucha información las peticiones pueden penalizarse en tiempo.

3. WebStorage

La recomendación para almacenar datos en el navegador pasa por este API, la cual permite almacenar pares clave-valor de una forma más intuitiva que con las cookies. La información almacenada no viajará en las peticiones al servidor como en el caso de las cookies, además el límite de almacenamiento aumenta hasta 5MB.

Básicamente tenemos dos propiedades en el objeto `Window` que podemos utilizar:

- `sessionStorage` mantiene un área de almacenamiento mientras dure la sesión de la página, independientemente de recargar la página.
- `localStorage` hace lo mismo, pero persiste incluso cuando el navegador se cierre y se reabra.

Ambas propiedades permiten crear un objeto `Storage` cuyo interfaz vamos a trabajar a continuación.

3.1. Gestionar datos en el almacén

Vamos a escribir una serie de datos en el almacén de sesión, utilizamos para ello el método `setItem()` pasando la clave y el valor asignado.

```
sessionStorage.setItem('dato1', 'valor1');
sessionStorage.setItem('dato2', 'valor2');
```

Para recuperar el valor utilizamos el método `getItem()`.

```
sessionStorage.getItem('dato1');
sessionStorage.getItem('dato2');
```

Los pares clave-valor estarán accesibles mientras la sesión esté abierta. Al cerrar la ventana del navegador los datos se borrarán.

3.2. Recorrer claves

La propiedad `length` nos da el número de claves almacenadas en el almacén. Con `key()` obtenemos el nombre de una clave en base a su índice.

```
for (let i = 0, key; i< sessionStorage.length; i++){
  key = sessionStorage.key(i);
  console.log(key + ': ' +sessionStorage.getItem(key));
}
```

3.3. Almacén local

El almacén `localStorage` es independiente de `sessionStorage`. Su contenido queda almacenado, aunque cerremos el navegador.

Vamos a guardar la fecha y la hora del último acceso, así como un contador de visitas.

```
let visitNum = localStorage.getItem('visitNum');
if(visitNum){
  let d = new Date (Number.parseInt(localStorage.getItem('lastAccess')));
  console.log('Última visita: ' + d.toLocaleString());
  console.log('Nº total de visitas: ' + visitNum);
  visitNum = Number.parseInt(visitNum);
  localStorage.setItem('visitNum', ++visitNum);
} else {
  console.log('Es la primera visita a la página');
  localStorage.setItem('visitNum', 1);
}
localStorage.setItem('lastAccess', new Date().getTime());
```

3.4. Borrar datos del almacén

Para borrar un dato del almacén tenemos el método `removeItem()`, para borrar todos los datos el método `clear()`. El siguiente fragmento de código borrar *dato1* de la sesión y vacía el almacén local.

```
sessionStorage.removeItem('dato1');
localStorage.clear();
```

3.5. Evento `storage`

Este evento se dispara cuando el almacén `localStorage` ha sido modificado en el contexto de otro documento, es decir, desde otra página diferente. El siguiente código muestra el manejador de eventos que reconoce la modificación del almacén. Entre las propiedades que podemos utilizar en el evento están:

- `key`: nombre de la clave que ha sido modificada.
- `oldValue`: Antiguo valor asignado.
- `newValue`: valor actualizado.
- `url`: Dirección desde la que se ha hecho la modificación.

Para probarlo abrimos un nuevo navegador y actualizamos `localStorage`.

```
window.addEventListener('storage', (event) => {
  if (event.key === 'visitNum'){
    alert('El número de visitas ha sido modificado.');
    $$result.clear();
    console.log('Valor antiguo: ' + event.oldValue);
    console.log('Nuevo valor: ' + event.newValue);
    console.log('url: ' + event.url);
  }
});
```