Fecha 06/05/2024
Subgrupo N. 1 y 2
Nota: [ ]

PROGRAMACION WEB - JAVASCRIPT PARTE III

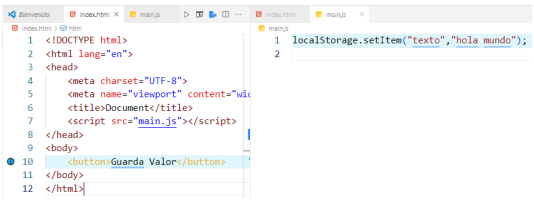
Abstract
El objeto Storage (API de almacenamiento web) permite almacenar datos de manera local en el navegador sin necesidad de realizar ninguna conexión con el servidor. De esta manera, cada cliente puede preservar información de la aplicación.

1. Storage

El navegador nos ofrece dos tipos de storage: localStorage y sessionStorage.

2. Local Storage

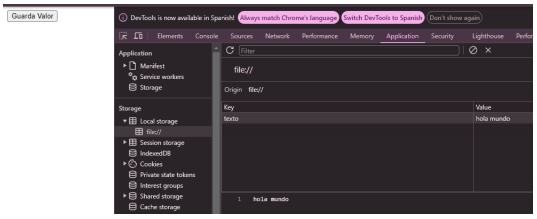
Los datos almacenados en localStorage (variable global preexistente) se almacenan en el navegador de forma indefinida (o hasta que se borren los datos de navegación del browser):



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="wi
    <title>Document</title>
    <script src="main.js"></script>
</head>
<body>
    <button>Guarda Valor</button>
</body>
</html>
```

```
localStorage.setItem("texto","hola mundo");
```

En el navegador ese código se vería así: debemos ir a ”Inspeccionar” después a ”Aplicaciones” y podemos ver cómo se ha guardado un espacio de memoria a través del navegador en donde está el localstorage con una clave y un valor.



Podemos así almacenar cualquier tipo de variables, en la imagen se muestra cómo se ha almacenado una variable adicional ”numero”, ”confirmar”, ”producto”

```
localStorage.setItem("texto","Hola mundo")
localStorage.setItem("numero",1243)
localStorage.setItem("confirmar",true)
localStorage.setItem("producto",{nombre:"heladera",precio:120000})
```

3. Localstorage:getitem

Podemos acceder a la información almacenada en localStorage utilizando getItem. Las claves y valores de Storage se guardan en formato de cadena de caracteres (DOMString).

```
})

localStorage.setItem("titulo","Computacion Maxi")
localStorage.setItem("numero",1243)
localStorage.setItem("confirmar",true)
localStorage.setItem("numeros",[2321,2313,2323,2323,258])
localStorage.setItem("producto",{nombre:"heladera",precio:120000})

const titulo = localStorage.getItem("titulo")
const contendorTitulo = document.querySelector("#contendorTitulo")
contendorTitulo.innerText = titulo

|
```

Cuando los datos se traen, si éstos son numéricos se hacen strings, por lo cual es necesario aplicar un cast para obtener números como tal, eso podemos hacerlo con `parseFloat(localStorage.getItem("numero")` de esta misma manera, con el resto de tipo de datos.

```
// getItem con string
const titulo = localStorage.getItem("titulo")
const contendorTitulo = document.querySelector("#contendorTitulo")
contendorTitulo.innerText = titulo

//getItem con number
const numero = parseFloat(localStorage.getItem("numero"))
console.log(typeof(numero))

//getItem con boolean
const confirmar = (localStorage.getItem("confirmar") === "true")
console.log(typeof(confirmar))

//getItem con array (primitivos)
const numeros = localStorage.getItem("numeros").split(",")
numeros.forEach((numero,index)=> numeros[index] = parseFloat(numero))
```

4. Recorriendo el storage

Es posible obtener todos los valores almacenados en localStorage o sessionStorage con un bucle.

Pero no podemos usar for...of porque no son objetos iterables, ni for...in porque obtenemos otras propiedades del objeto que no son valores almacenados.

```
//Ciclo para recorrer las claves almacenadas en el objeto
localStorage
for (let i = 0; i < localStorage.length; i++) {
    let clave = localStorage.key(i);
    console.log("Clave: "+ clave);
    console.log("Valor: "+ localStorage.getItem(clave));
}
```

5. Eliminando el storage

Podemos eliminar la información almacenada en sessionStorage o localStorage usando el método removeItem o clear:

```
localStorage.setItem('bienvenida', '¡Hola Programacion Web!');
sessionStorage.setItem('esValido', true);

localStorage.removeItem('bienvenida');
sessionStorage.removeItem('esValido');
localStorage.clear()    //elimina toda la información
sessionStorage.clear() //elimina toda la información
```

Para saber cuántos objetos tengo en el storage utilizo `localStorage.length`

// sin apertura y cierre de paréntesis al último de la línea

6. Session storage

La propiedad sessionStorage permite acceder a un objeto Storage asociado a la sesión actual. La propiedad sessionStorage es similar a localStorage, la única diferencia es que la información almacenada en localStorage no posee tiempo de expiración, por el contrario la información almacenada en sessionStorage es eliminada al finalizar la sesion de la página. La sesión de la página perdura mientras el navegador se encuentra abierto, y se mantiene por sobre las recargas y reaperturas de la página. Abrir una página en una nueva pestaña o ventana iniciará una nueva sesión, lo que difiere en la forma en que trabajan las cookies de sesión.

7. JSON

JavaScript Object Notation (JSON) es un formato basado en texto plano, para representar datos estructurados con la sintaxis de objetos de JavaScript. Es comúnmente utilizado para enviar y almacenar datos en aplicaciones web.

Aunque es muy parecido (casi similar) a la sintaxis de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; parsear) y generar JSON. JSON es un string con un formato específico.

8. Conversiones de/hacia JSON

Cuando sea necesario enviar un objeto Javascript al servidor o almacenarlo en storage, será necesario convertirlo a un JSON (una cadena) antes de ser enviado.

Stringify

Con JSON.stringify podemos transformar un objeto JavaScript a un string en formato JSON.

```
const producto1 = { id: 2, producto: "Arroz" };
const enJSON    = JSON.stringify(producto1);

console.log(enJSON); // {"id":2,"producto":"Arroz"}
console.log(typeof producto1); // object
console.log(typeof enJSON);    // string

localStorage.setItem("producto1", enJSON);
// Se guarda {"id":2,"producto":"Arroz"}
```

Parse

Con JSON.parse podemos transformar string en formato JSON a objeto JavaScript.

```
const enJSON    = '{"id":2,"producto":"Arroz"}';
const producto1 = JSON.parse(enJSON);

console.log(typeof enJSON);    // string
console.log(typeof producto1); // object
console.log(producto1.producto); // Arroz

const producto2 = JSON.parse(localStorage.getItem("producto1"));
console.log(producto2.id);  // 2
```

OTROS EJEMPLOS

```
13  const producto={nombre:"heladera", precio:120000}    "produ
14  localStorage.setItem("producto", JSON.stringify(producto));
```

Otra forma de hacerlo directamente es la siguiente

```
39  const productoJson = '{"nombre":"heladera","precio":120000}'
40  localStorage.setItem("producto",productoJson)
```

La diferencia es que en la segunda imagen se utilizan comilla simple dentro del string del JSON y luego entre doble comilla cada clave, SIN EMBARGO ESTO NO ES RECOMENDADO.

Finalmente, si queremeos recuperar el objeto, lo que se hace es lo siguiente:

```
//getItem con objetos
const productoRecupera = JSON.parse(localStorage.getItem("producto"))
console.log(productoRecupera.nombre)
```

JavaScript tiene un tipado dinámico, eso es necesario entender. TypeScript si tiene tipado de datos; se recomienda revisar TypeScript.

9. EJEMPLO APLICADO

En el siguiente ejemplo almacenaremos una array de objetos:

```
const productos = [{ id: 1,  producto: "Arroz", precio: 125 },
                   { id: 2,  producto: "Fideo", precio: 70 },
                   { id: 3,  producto: "Pan"  , precio: 50},
                   { id: 4,  producto: "Flan" , precio: 100}];

const guardarLocal = (clave, valor) => { localStorage.setItem(clave, valor) };
//Almacenar producto por producto
for (const producto of productos) {
    guardarLocal(producto.id, JSON.stringify(producto));
}
// o almacenar array completo
guardarLocal("listaProductos", JSON.stringify(productos));
```

10. EJERCICIO

Hacer un ejercicio de investigación que permita utilizar FETCH y utilizar un archivo de JSON para operar.

Vamos a hacer un ”Mantenimiento” completo de personas utilizando localStorage y vamos a ir almacenando en un archivo JSON todas las personas que se ingresen. Probar lo mismo con Session Storage y ver las diferencias.

Además haremos una página que permita recuperar todas las personas ingresadas.