JAVASCRIPT 06

DATENSPEICHERUNG:
COOKIES & LOCALSTORAGE

Stark eingeschränkte Möglichkeiten für die Datenspeicherung

- JS läuft aus Sicherheitsgründen in einer Sandbox = Programme können nur auf Ressourcen zurückgreifen können die ihnen der Browser explizit zur Verfügung stellt
- Dateisystem kann nicht verwendet werden
- Clientseitig – Daten werden nicht zum Server geschickt
- Probleme der vorhandenen Möglichkeiten
  - Datenmenge begrenzt
  - Nur string-Werte verarbeiten (Lösung: mit JSON arbeiten)
  - Mangel an Sicherheit -> Daten können ohne große Probleme ausgewertet werden
  - **(keine sensible Daten darin abspeichern!!)**

Möglichkeiten der Datenspeicherung über JS

COOKIES

Nutzlast
max. 4KB

max. Lebenszeit
praktisch unbegrenzt

Geltungsbereich
Alle Browserfenster / Tabs

Haltbarkeitsdatum wird bei der Erzeugung des Cookies festgelegt

SESSION STORAGE

Nutzlast
5 bis 10MB

max. Lebenszeit
bis Seite geschlossen wird

Geltungsbereich
ein individuelles Browserfenster / Tab

Wird beim Schließen des Browserfensters automatisch gelöscht

LOCAL STORAGE

Nutzlast
5 bis 10MB

max. Lebenszeit
praktisch unbegrenzt

Geltungsbereich
Alle Browserfenster / Tabs

Wird nur von Javascript oder Löschen des Browser-Cache gelöscht

INDEXEDDB

Nutzlast
250MB

max. Lebenszeit
praktisch unbegrenzt

Geltungsbereich
Alle Browserfenster / Tabs

Wird nur von Javascript oder Löschen des Browser-Cache gelöscht

Cookies verwenden

- Hinweis: keine Cookies von lokalen Seiten
- Key-Value Speicher
- Ohne Pfad setzt der Browser das Cookie immer für die aktuelle Seite
- **Anwendungsfall**: Login, Warenkorb

```
// Cookies unter document.cookie erreichbar
// neuen Wert zuweisen: Name des Cookies deklarieren und String setzen
document.cookie = "meinCookie=Hier steht ein beliebiger Wert.";
```

- Cookie in einer Funktion erstellen mit Ablaufdatum

```
function setCookie(cookieName, inhalt, dauer) {
    // neues Objekt vom Typ Date
    let datum = new Date();
    // gewünschtes Ablaufdatum festlegen
    // setTime ermöglicht es, einen neuen Zeitpunkt vorzugeben
    // getTime() = aktuelles Datum plus gewünschte Dauer in Millisekunden
    // Da Dauer in Tagen sinnvoller => 24 h * 60 min * 60 sec * 1000 => Wert in Millisekunden
    datum.setTime(datum.getTime() + (dauer*24*60*60*1000));
    // für gewünschtes Format toGMTString() Methode
    // für Ablaufdatum expires vor Methode stellen
    let ablaufdatum = "expires=" + datum.toGMTString();
    document.cookie = cookieName + "=" + inhalt + ";" + ablaufdatum;
}
```

COOKIE ABFRAGEN

```
function getCookie(cookieName) {
    cookieName += "=";
    /* um Inhalt des Cookie zu ermitteln = Eigenschaft document.cookie
     * um ev. vorhandene Umlaute oder Sonderzeichen richtig darzustellen Methode decodeURIComponent()
     * man erhält kompletten Cookie String mit Namen, Inhalt, Ablaufdatum und Pfadnamen falls vorhanden */
    let decCookie = decodeURIComponent(document.cookie);
    /* kompletten Cookie String in einer Zeichenkette
     * Methode split(';') zerteilt Zeichenkette und erzeugt ein Array dessen Felder die Inhalte
     * Der Bereiche zwischen den Semikolons enthalten */
    let arr = decCookie.split(";");

    for (let i = 0; i < arr.length; i++) {
        let inhalt = arr[i];
        /* Für Überprüfung hilfreich, alle Leerzeichen, die eventuell am
         * Anfang des Array-Feldes vorhanden sind zu entfernen mit while und substring()-Methode */
        while (inhalt.charAt(0) == ' ') {
            inhalt = inhalt.substring(1);
        }
        /* Wenn Variable mit cookieName existiert => Name des Cookies
         * Überprüfung mit indexOf() Methode ob dieser Ausruck zu Beginn der aktuellen
         * Zeichenkette steht */
        if(inhalt.indexOf(cookieName) == 0) {
            /* Wenn ja, gewünschten Inhalt mit substring() Methode extrahieren
             * Da Name des Cookies nicht zurückgegeben werden soll, wird als Startpunkt cookieName.length gewählt */
            return inhalt.substring(cookieName.length);
        }
    }
    return "";
}
```

ANWENDUNGSBEISPIEL

- Der User gibt seinen Namen beim Laden der Seite ein z.B. MAX
- Der Name wird 180 Tage gespeichert
- Innerhalb dieser 180 Tage wird der User mit „Hallo MAX“ begrüßt

```
<body onload="checkCookie()">
<button type="button" onclick="setCookie('anwender','',-1)">Cookie
löschen</button>
<script>
    function setCookie(cookieName, inhalt, dauer) {
        let datum = new Date();
        datum.setTime(datum.getTime() + (dauer*24*60*60*1000));
        let ablaufdatum = "expires=" + datum.toGMTString();
        document.cookie = cookieName + "=" + inhalt + ";"
                        + ablaufdatum;
    }
    function getCookie(cookieName) {
        cookieName += "=";
        let decCookie = decodeURIComponent(document.cookie);
        let arr = decCookie.split(";");
        for (let i = 0; i < arr.length; i++) {
            let inhalt = arr[i];
            while (inhalt.charAt(0) == ' ') {
                inhalt = inhalt.substring(1);
            }
            if(inhalt.indexOf(cookieName) == 0) {
                return inhalt.substring(cookieName.length);
            }
        }
        return "";
    }
    function checkCookie() {
        let anwender = getCookie("anwender");
        if(anwender != "") {
            alert("Hallo " + anwender + "!");
        } else {
            anwender = prompt("Gib deinen Namen ein:");
            alert("Hallo " + anwender + "!");
            if (anwender != "" && anwender != null) {
                getCookie("anwender", anwender, 180);
            }
        }
    }
</script>
</body>
```

Session Storage

- Key-Value Speicher
- Existiert nur im Tab im aktuellen Browser
- Einfache Anwendung

- **Anwendungsfall**: Sprachauswahl speichern

```
// setzen eines Key-Value-Pairs
sessionStorage.setItem("key", "value");

// gespeicherte Daten holen
sessionStorage.getItem("key");

// löschen der Daten
sessionStorage.removeItem("key");
```

Daten mit localStorage speichern

- Key-Value Speicher der die Werte als String speichert
- Einfache Anwendung

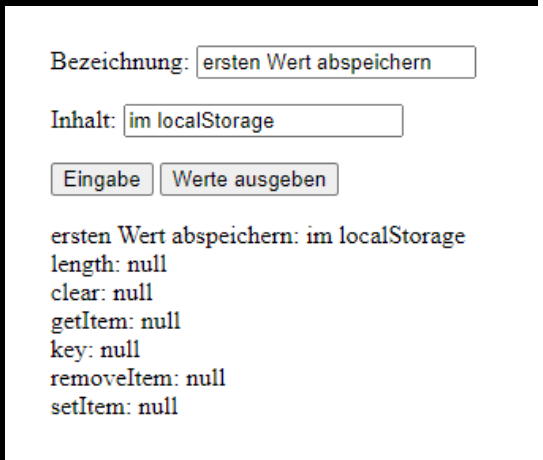
- **Anwendungsfall**: Userbezogene Daten speichern

```
// setzen eines Key-Value-Pairs
localStorage.setItem("key", "value");

// gespeicherte Daten holen
localStorage.getItem("key");

// löschen der Daten
localStorage.removeItem("key");
```

Daten mit localStorage speichern

- Mit `localStorage.removeItem(wert);` wird ein spezifischer Eintrag gelöscht
- Ist ein Objekt, deshalb kann für Ausgabe von allen Elementen eine for-in-Schleife verwendet werden

```
<body>
<p>Bezeichnung: <input id="bezeichnung" value=""></p>
<p>Inhalt: <input id="inhalt" value=""></p>
<button type="button" onclick="speichern()">Eingabe</button>
<button type="button" onclick="ausgeben()">Werte ausgeben</button>
<button type="button" onclick="loeschen()">Werte im localStorage
löschen</button>
<p id="pAusgabe"></p>
<script>
    function speichern() {
        let bez = bezeichnung.value;
        let inh = inhalt.value;
        // Wert wird im localStorage gespeichert
        localStorage.setItem(bez, inh);
    }
    function ausgeben() {
        let ausgabe = "";
        let i = 0;
        for (let wert in localStorage) {
            // mit getItem() wird der Wert aus
            // dem localStorage abgerufen
            ausgabe += wert + ": " + localStorage.getItem(wert);
            ausgabe += "<br>";
            i++;
            // Um weitere Attribute und Methoden des Objektes nicht
            // auszugeben = zusätzlicher Zähler der die Länge des
            // Eintrages überprüft
            if(i == localStorage.length) {
                break;
            }
        }
        pAusgabe.innerHTML = ausgabe;
    }
    function loeschen() {
        // Werte werden gelöscht
        localStorage.clear()
    }
</script>
</body>
```



IndexedDB

- Kann Objekte und Key-Value-Pairs speichern
- Asynchron (im Gegensatz zu Session und Local Storage)
  - Durch ein Event getriggert
- SQL Statements möglich

- **Anwendungsfall**: große Menge von Objekte speichern - WebApps

```
// Kundendaten
const customerData = [
  { cnumber: "17844", lastname: "Maier", firstname: "Laura", email: "laura@maier.at" },
  { cnumber: "17845", lastname: "Huber", firstname: "Franz", email: "franz@huber.at" }
];

// gespeicherte Daten holen
const customerDB = "Customer DB";

// Datenbank Öffnen, 2 Parameter ist die Version der DB
let request = indexedDB.open(customerDB, 1);

request.onerror = function(event) {
   // Error behandeln
}
request.onupgradeneeded = function(event) {
   // Datenbank holen
   let db = event.target.result;

   //ObjectStore erstellen wo die Daten gespeichert werden, cnumber ist der primary-key
   let objectStore = db.createObjectstore("Customers", {keyPath: "cnumber" });

   // Werte speichern
   for(let i in customerData) {
      objectStore.add(customerData[i]);
   }
}
```

AUFGABE

1. Erstelle eine Seite, die beim Laden überprüft, ob Cookies mit dem Namen und der Lieblingsfarbe des Besuchers vorhanden sind. Trifft dies zu, soll sie eine entsprechende Meldung auf der Seite ausgeben. Ist das nicht der Fall, soll das Programm mit zwei prompt-Befehlen die entsprechenden Werte abfragen. Speichere diese daraufhin in zwei Cookies.
2. Gestalte eine Seite, die genau die gleiche Funktion wie das Programm oben hat. Verwende jedoch localStorage für die Datenspeicherung

- Anmerkung für Übung
  - Wenn man bei getItem()-Methode einen Bezeichner eingibt, der nicht vorhanden ist, gibt dieser den Wert null zurück – und keine leere Zeichenkette wie bei der Funktion für die Cookies. Daher muss man die Bedingung bei der Überprüfung dementsprechend anpassen.

ENDE

QUELLE: JAVASCRIPT
PROGRAMMIEREN FÜR EINSTEIGER
ISBN: 978-3-96645-016-4