

Data Management on the Client Side



Local Storage



IndexedDB

Outline

1. Web Storage API
2. IndexedDB

Web Storage API



Local Storage

Web Storage API

- The Web Storage API provides mechanisms to **store key/value pairs** locally within the user's browser
- The Web storage limit is at least 5MB and information is never transferred to the server
- Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.
- It provides two objects for storing data on the client:
 - **localStorage** - stores data with no expiration date
 - **sessionStorage** - stores data for one session (data is lost when the browser tab is closed)

The localStorage Object

- The localStorage object stores the data with no expiration date. The data will not be deleted when the browser tab is closed.

```
// Store
localStorage.setItem("lastname", "Saleh");
// Retrieve
console.log( localStorage.getItem("lastname") );
```

- The example above could also be written like this:

```
// Store
localStorage.lastname = "Saleh";
// Retrieve
console.log( localStorage.lastname );
```

- The syntax for removing the "lastname" localStorage item is as follows:

```
delete localStorage.lastname;
```

Note:

Name/value pairs are always stored as strings. Remember to convert them to desired format!

localStorage Example

- Store the number of times a user has clicked a button
 - clickCount is converted to a number to be able to increase the counter

```
function clickCounter() {  
  if (localStorage.clickCount) {  
    localStorage.clickCount = parseInt(localStorage.clickCount) + 1;  
  } else {  
    localStorage.clickCount = 1;  
  }  
  document.querySelector("#count").innerHTML = `Button clicked  
    ${localStorage.clickCount} times.`;  
}
```

sessionStorage Object

- The **sessionStorage** object is the same as the localStorage object, except that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.
- The following example counts the number of times a user has clicked a button, in the current session

```
function clickCounter() {  
  if (sessionStorage.clickCount) {  
    sessionStorage.clickCount = parseInt(sessionStorage.clickCount) + 1;  
  } else {  
    sessionStorage.clickCount = 1;  
  }  
  document.querySelector("#count").innerHTML = `Button clicked  
    ${sessionStorage.clickCount} times.`;  
}
```



IndexedDB

IndexedDB

- IndexedDB is a database built into the browser and allow storing and querying JSON documents.
- IndexedDB database is a collection of Object Stores
- Object Store: stores a **collection** of objects, similar to tables in a relation database.
- Object: could be a json object.
- Its API is bit complicated hence IDB library <https://github.com/jakearchibald/idb> could be used to simply interacting with IndexedDB
 - IDB library is an async/await **wrapper** for IndexedDB

IndexedDB

- Objects can arrange in object stores
 - Objects in a store usually **have similar purpose** but they may have slightly different schema
 - Objects within a Store have unique identifiers (primary key)

Database

objectStore

key1: value1
key2: value2
key3: value3
key4: value4
key5: value5

objectStore

key1: value1
key2: value2
key3: value3
key4: value4
key5: value5

objectStore

key1: value1
key2: value2
key3: value3
key4: value4
key5: value5

Open and Created Database and ObjectStore

```
import { openDB } from 'https://unpkg.com/idb?module';
const dbName = 'heroes-db';    // database name
const dbVersion = 1;    // database version (not IndexedDB version)
const heroesStoreName = 'heroes';    // Name of the objects store
const db = await openDB(dbName, dbVersion, {
    // This callback only runs ONE time per database version.
    // Use it to create object stores.
    upgrade(db) {
        if (!db.objectStoreNames.contains(heroesStoreName)) {
            /* keyPath: specify the primary key for each object in the object store.
               Set autoIncrement to true if you want IndexedDB to handle primary
               key generation for you */
            db.createObjectStore(heroesStoreName, {
                keyPath: 'id', autoIncrement: true,
            });
        }
    },
},
);
```

openDB method

- The **upgrade** callback passed to openDB only runs ONE time per database version
 - use db.**createObjectStore** to create object stores
 - **keyPath**: specify the primary key for each object in the object store
 - Set **autoIncrement** to true if you want IndexedDB to handle primary key generation
 - Use it store.**createIndex** to create an index to speed-up queries by properties other than the primary key
 - In IndexedDB, an index is just another "shadow store" that's based off the main store
 - Adding an index is like creating the same store with a different 'keyPath'
 - Just like how the main store is auto sorted by the primary key, the index store is auto sorted by the index key

CRUD Operations

// Get all

```
await db.getAll(heroesStoreName);
```

// Get object by id

```
await db.get(heroesStoreName, heroId);
```

// Add object

```
await db.add(heroesStoreName, hero);
```

// Update object

```
await db.put(heroesStoreName, hero);
```

// Delete object

```
await db.delete(heroesStoreName, heroId);
```

Query using an Index

- Index allows one to query on properties other than the primary key
- Use the index to return all the objects where the index value matches a parameter

// Get all using the index to return objects where the index value = heroType
`await db.getAllFromIndex(heroesStoreName, 'heroTypeIndex', heroType);`

// Get a single object using the index to return the object where the index value = studentId
`await db.getFromIndex(studentsStoreName, 'studentIdIndex', studentId);`

Query by Range

- Whenever you call `.get()` or `.getAll()`, you can substitute the key with a **range**, whether querying by a primary key or index key.
 - `const goodRange = IDBKeyRange.lowerBound(3);`
query where `key >= value`
 - `const midRange = IDBKeyRange.bound(2.01, 2.99);`
query where `lowerBound <= key <= upperBound`
 - `const weakRange = IDBKeyRange.upperBound(2);`
query where `key <= upperBound`
 - Range works on strings too, e.g.,
`IDBKeyRange.bound('ae1234', 'ss6777')`

More info @ <https://developer.mozilla.org/en-US/docs/Web/API/IDBKeyRange>

Resources

- Web Storage API

[https://developer.mozilla.org/en-US/docs/Web/API/Web Storage API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)

- IndexedDB

<https://javascript.info/indexeddb>

[https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB API](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API)