Day 17 - HTML5 Web Storage

HTML5 Web Storage

Web Storage (sessionStorage and localStorage) is an HTML5 API that offers important benefits over traditional cookies. Prior to HTML5, app data had to be stored in cookies and included in every server request.

Web storage is more secure and large amounts of data can be stored locally without affecting performance. Data storage limit of cookies in many browsers is 4 KB per cookies, while web storages can store far larger data (at least 5MB) and never be transferred to the server. All sites from the same or one origin can store and access the same data.

Data being stored can be accessed using JavaScript, which allows you to leverage client-side scripting to do many things that’ve traditionally involved server-side programming and relational databases. There are two Web Storage objects:

- sessionStorage
- localStorage

localStorage is similar to sessionStorage, except data store in sessionStorage gets cleared when the page session ends and data stored in localStorage has no expiration time.

Data stored in either localStorage or sessionStorage is specific to the protocol of the page. The keys and values are always strings (integer keys will be automatically converted to strings).

sessionStorage

Only available within the browser tab or window session. It’s designed to store data in a single web page session, which means if the window is closed, the session data will be removed. Both sessionStorage and localStorage have similar methods.

localStorage

Used to store data on the browser with no expiration. Data will be available on the browser even after the browser is closed and is kept even between browser sessions. This means data is still available when the browser is closed and reopened, and also instantly between tabs and windows.

Both types of Web Storage data are not available between different browsers. For example, storage objects created in Firefox can't be accessed in Internet Explorer, just like cookies. There's five methods for working on local storage: *setItem(), getItem(), removeItem(), clear(), key().*

Use case of Web Storages

- Store data temporarily
- Save products that users placed in their shopping cart
- Data can be made available between page requests, multiple browser tabs, and between browser sessions using localStorage
- Can be used offline completely using localStorage.
- Can be a great performance win when some static data is stored on the client to minimize the number of subsequent requests. Even images can be stored in strings using Base64 encoding.
- Can be used for user authentication

Most above examples make sense to use localStorage. As for session storage, it's useful for when we want to get rid of data as soon as the window is closed, or if we don't want the application to interfere with the same application that's open in another window.

HTML5 Web Storage Objects

HTML web storage provides two objects for storing data on the client:

- window.localStorage - stores data with no expiration date
- window.sessionStorage - Stores data for one session. Most modern browsers support Web Storage, but it's a good idea to check browser support for localStorage and sessionStorage.

Available methods for Web Storage objects:

- *localStorage* - displays the localStorage object.
- *localStorage.clear()* - To remove everything in the local storage
- *localStorage.setItem()* - to store data in localStorage. Takes key and value parameters.
- *localStorage.getItem()* - Displays data stored in the localStorage. Takes a key as a parameter.
- *localStorage.removeItem()* - to remove stored item from localStorage. Takes a key as a parameter.
- *localStorage.key()* - display a data stored in localStorage. Takes an index as a parameter.

Setting item to localStorage

Data that is stored in localStorage will be stored as a string. If storing an array or object, we should stringify it first to keep the format. Otherwise, we lose the array structure or the object structure of the original data.

```
localStorage.setItem('key', 'value')
```

Storing string in localStorage:

```
localStorage.setItem('firstName', 'Chris') // Since the value is a string, we don't need to stringify it
console.log(localStorage)

//log:
Storage {firstName: 'Chris, length: 1}
```

Storing a number in local storage:

```
localStorage.setItem('age', 24)
console.log(localStorage)

//log:
Storage {age: '24', firstName: 'Chris', length: 2}
```

Storing an array in localStorage. If storing an array, an object, or an object array, we should stringify it first.

```
const skills = ['HTML', 'CSS', 'JS', 'React']
const skillsJSON = JSON.stringify(skills, undefined, 4)
localStorage.setItem('skills', skillsJSON)
console.log(localStorage)

//log:
Storage {age: '200', firstName: 'Asabeneh', skills: 'HTML,CSS,JS,React', length: 3}
```

```
let skills = [
  { tech: 'HTML', level: 10 },
  { tech: 'CSS', level: 9 },
  { tech: 'JS', level: 8 },
  { tech: 'React', level: 9 },
  { tech: 'Redux', level: 10 },
  { tech: 'Node', level: 8 },
  { tech: 'MongoDB', level: 8 }
]

let skillJSON = JSON.stringify(skills)
localStorage.setItem('skills', skillJSON)
```

Storing an object in localStorage. Before we store objects to localStorage, the object has to be stringified.

```
const user = {
  firstName: 'Chris',
  age: 250,
  skills: ['HTML', 'CSS', 'JS', 'React']
}

const userText = JSON.stringify(user, undefined, 4)
localStorage.setItem('user', userText)
```

Getting item from localStorage

```
localStorage.getItem('key')
```

```
let firstName = localStorage.getItem('firstName')
let age = localStorage.getItem('age')
let skills = localStorage.getItem('skills')
console.log(firstName, age, skills)

//log:
"Chris", "24", "['HTML','CSS','JS','React']"
```

Our skills array is in string format, so we should use JSON.parse() to parse it to a normal array.

```
let skills = localStorage.getItem('skills')
let skillsObj = JSON.parse(skills, undefined, 4)
console.log(skillsObj)

//log:
['HTML','CSS','JS','React']
```

Clearing localStorage

Clears everything in localStorage

```
localStorage.clear()
```