Web Storage API

- localStorage and sessionStorage are read only properties on the window object.
- `window.hasOwnProperty("sessionStorage")` returns `true`
- Both localStorage and sessionStorage are objects.

```
typeof localStorage //object
typeof sessionStorage //object
```

- Both localStorage and sessionStorage follow same origin policy.

Same Origin Policy

- Let's see same origin policy in short:
- An origin consists of three things, **1) protocol, 2)host 3)port**

- Any change in path is not considered as a different origin.
- Suppose we have a website link **http://www.abc.com**
- **https**://www.abc.com would be a **different** origin, since they have different **protocols**
- http://**drive**.abc.com would also be a **different** origin, since it has different host(different subdomain===different host)
- http://www.abc.com:**81** would be a **different** origin, since they have different ports HTTP by default hits at port 80.
- http://www.abc.com/**about.html** is **same** origin, since they only differ by **path**.

- Why same origin policy is important? Because storage created on one origin, can't be accessed on some other origin.
- For example you have executed the following code on **http**://www.abc.com

```
localStorage.setItem("animal","bear");
```

- And you try to access this stored value on **https**://www.abc.com

```
localStorage.getItem("animal");
```

- It will return `null`, since these two websites represent different origins, and on the https version of the website , these is no such key as `animal`.

Important Methods and Properties of Web Storage

- **setItem**

- Takes two arguments, key and value
- Both these arguments must be key. Otherwise unexpected things might happen.
- If no similar key exists, creates a new key with the specified value.
- If key exists, it overwrites the value for that key.

```
localStorage.setItem("animal","bear");
```

- We can also add a key directly, without using setItem

```
localStorage.animal="bear"
```

- We can also use array notation to set the keys

```
localStorage["animal"]="bear"
```

- If you wish to store an array or object, instead of simple string value. First serialize it using JSON.stringify().

```
let animals = ['lion','bear','fox']
localStorage.setItem("animals",JSON.stringify(animals))
```

- But why? Because setItem always tries to convert the value to a string, before storing.
- But if we try to convert an object to a string it will give `[object Object]`
- We don't want to store `[object Object]`, we want to store the actual object. For that we do serialization using JSON.stringify().

- **getItem**
  - It takes only one argument, the key for which you want to get the value.

```
localStorage.getItem("animal") //returns "bear"
```

- If no such key exists, returns `null`.
- Similar to setItem we can use dot and array notation on getItem as well.

```
localStorage.animal //returns bear
localStorage["animal"] //returns bear
```

- If we are using the above mentioned methods, and if no key exists, it returns `undefined`.

- **removeItem**
  - Takes only one argument, the key to be removed.

```
localStorage.removeItem("animal")
```

  - An alternative to this is using the delete operator

```
delete localStorage.animal
```

- **clear**
  - Takes no argument
  - Removes all the keys.

```
localStorage.clear() //empties the localStorage
```

- **length**
  - This is a property, it gives the number of elements stored.

```
localStorage.setItem("animal","bear");
localStorage.setItem("flower","sunflower");
localStorage.length//2
```

- **key**

- This method returns the **key** at a particular index.
- The order depends on the user-agent(browser). So don't rely on it.
- Listing all the keys present in the localStorage

```
for(let i=0;i<localStorage.length;i++){
  console.log(localStorage.key(i);
}
```

- We can also use for-in loop to get all the keys

```
for(key in localStorage){
  if(localStorage.hasOwnProperty(key)){
    console.log(key)
  }
}

//we used hasOwnProperty since the in keyword searches
//in the whole proptotype chain
```

- Printing all the values

```
for(let i=0;i<localStorage.length;i++){
  console.log(localStorage.getItem(localStorage.key(i)))
```

- It returns null if no key exists at that key.

Local vs Session Storage

Cookies vs. Local Storage vs. Session Storage

	Cookies	Local Storage	Session Storage
Capacity	4kb	10mb	5mb
Browsers	HTML4 / HTML 5	HTML 5	HTML 5
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage Location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

- Local and session storage were introduced with HTML5. Before that cookies were used to store user related data.
- localStorage and sessionStorage are stored in a browser. Data stored in one browser can’t be accessed from the other browser.
- Unlike cookies, that data in local or session storage is not sent to the server with each request.
- Data like user preferences should be stored in the local storage, rather than on the cookie. This saves the bandwidth as well as gives greater speed to our web-app.
- local and sessionStorage save data around 5MB. While cookies can store upto 4kb only.
- Some browsers can allow unlimited data to be stored in the session storage.
- Both localStorage and sessionStorage stored in incognito mode are cleared when the incognito window is closed.

Local Storage

- Different origins have different session and localStorage.
- Data stored in the local storage doesn’t expire.
- The data must be cleared manually.
- It can persist page reloads and even the closing of the browser.
- Data created for an origin(refer to same origin policy mentioned above) can be accessed in different tabs that are having the same origin.
- The size depends on the device and browser you are using, but generally it is close to 5MB.

Session Storage

- When a page is loaded in a tab, a **session** starts.
- It is similar to localStorage in many aspects.
- The data can persist page reloads.
- Data stored in session storage gets cleared when the browser or its tab is closed.
- Each tab has its own different sessionStorage even if the origin is same.
- The size depends on the device and browser you are using, but generally it is close to 5MB.

Miscellaneous

- You get QUOTA_EXCEDDED_ERR if you try to insert data into an already full localStorage.

- Calculating the size of localStorage

```
let localStorageSize = function () {

   let _lsTotal = 0,_xLen, _x;

   for (_x in localStorage) {
    //if the key is not present in localStorage skip it.
   if (!localStorage.hasOwnProperty(_x)) continue;
        //if present, add the length of each key(it is a string)
        //string is JS take 2 Bytes
       _xLen = (localStorage[_x].length + _x.length) * 2;
       _lsTotal += _xLen;
   }
  //divide by 1024 to get number of kilobytes
 return  (_lsTotal / 1024).toFixed(2);
}

//retuns the size in kb
```

- Web storage data is saved in an SQLite file in a subdirectory in the user's profile in Google Chrome. On Windows PCs, the subdirectory is stored at **\AppData\Local\Google\Chrome\User Data\Default\Local Storage.**

References

- https://www.youtube.com/watch?v=MOd5cTJ6kaA&t=484s(video by Akshay Saini)
- https://www.youtube.com/watch?v=GihQAC1I39Q(video by Web Dev Simplified)
- https://dev.to/therajatg/local-storage-and-session-storage-javascript-i10(article by Rajat Gupta)
- https://stackoverflow.com/a/15841014/16716190(answer by Robert Harvey)

- https://www.youtube.com/watch?v=xSv-9Yod83Q(video by Code With Harry)
- https://dev.to/alexdevero/getting-started-with-web-storage-api-local-storage-and-session-storage-2o8f(article by Alex Devero)
- https://dev.to/gyanendraknojiya/javascript-all-about-local-storage-session-storage-and-cookies-f47(an article by Gyanendra Kumar Knojiya)