Cookies and Web Storage

As we've learned in previous sections, most data on the Web is stored in a database on a server, and we use the browser to retrieve this data. However, sometimes data is stored locally for the purposes of persisting throughout an entire session or until a specified expiration date.

In this reading, we'll go over using **cookies**to store data versus using the**Web Storage API**and the use cases for each storage method.

Cookies

Cookies have been around forever, and they are still a widely used method to store information about a site's users.

What is a cookie?

A cookie is a small file stored on a user's computer that holds a bite-sized amount of data, under 4KB. Cookies are included with HTTP requests. The server sends the data to a browser, where it's typically stored and then sent back to the server on the next request.

What are cookies used for?

Cookies are used to store stateful information about a user, such as their personal information, their browser habits or history, or form input information they have filled out. A common use case for cookies is storing a *session cookie*on user login/validation. Session cookies are lost once the browser window is closed. To make sure the cookie persists beyond the end of the session, you could set up a *persistent cookie*with a specified expiration date. A use case for a persistent cookie is an e-commerce website that tracks a user's browsing or buying habits.

How to create a cookie in Javascript:

As we've previously covered, the `document`interface represents the web page loaded in a user's browser. Since cookies are stored on a user's browser, it makes sense that the `document`object also allows us to get/set cookies on a user's browser:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
const secondCookie = "favoriteDog=bambi";
document.cookie = secondCookie;
document.cookie; // Returns "favoriteCat=million; favoriteDog=bambi"
```

Using the following syntax will create a new cookie:

```
document.cookie = aNewCookieHere;
```

If you want to set a second cookie, you would assign a new key value pair using the same syntax a second time. Make sure to set the cookie to a string formatted like a key-value pair:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
document.cookie; // Returns "favoriteCat=million"
```

Formatting your string like we do in the `firstCookie`variable above sets the cookie `value`with a defined key, known as the cookie's `name`, instead of an empty `name`. Refer to the MDN docs on Document.cookiefor more examples.

You can view all the cookies a website is storing about you by using the Developer Tools. On **Google Chrome**, see the **Application tab**, and on**Firefox**, see the **Storage tab**.

Deleting a cookie:

We can delete our own cookies using JavaScript by setting a cookie's expiration date to a date in the past, causing them to expire:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
document.cookie; // Returns "favoriteCat=million"

// specify the cookies "name" (the key) with an "=" and set the  expiration
// date to the past
document.cookie = "favoriteCat=; expires = Thu, 01 Jan 1970 00:00:00 GMT";
document.cookie; // ""
```

We can also delete cookies using the Developer Tools!

Navigate to a website, such as Amazon, and add an item to your cart. Open up the Developer Tools in your browser and delete all the cookies. In Chrome, you can delete cookies by highlighting a cookie and clicking the delete button. In Firefox, you can right-click and delete a cookie. If you've deleted all the cookies in your Amazon cart, and you refresh the page, you should notice your cart is now empty.

Web Storage API

Cookies used to be the only way to store data in the browser, but with HTML5 developers gained access to the Web Storage API, which includes**localStorage**and **Session Storage**. Here are the differences between the two, according to MDN:

`sessionStorage`:

- Stores data only for a *session*, or until the browser window or tab is closed
- Never transfers data to the server
- Has a storage limit of 5MB (much larger than a cookie)

The following example from MDNshows how we can use sessionStorage to autosave the contents of a text field and restore the contents of that text field if the browser is accidentally refreshed.

```
// Get the text field that we're going to track
let field = document.getElementById("field");

// See if we have an autosave value
// (this will only happen if the page is accidentally refreshed)
if (sessionStorage.getItem("autosave")) {
  // Restore the contents of the text field
  field.value = sessionStorage.getItem("autosave");
}

// Listen for changes in the text field
field.addEventListener("change", function () {
  // And save the results into the session storage object
  sessionStorage.setItem("autosave", field.value);
});
```

`localStorage`:

- Stores data with no expiration date and is deleted when clearing the browser cache
- Has the maximum storage limit in the browser (much larger than a cookie)

Like with `sessionStorage`, we can use the `getItem()`and `setItem()`methods to retrieve and set `localStorage`data. The following example from MDNwill:

- Check whether `localStorage`contains a data item called `bgcolor`using`getItem()`.
- If `localStorage`contains `bgcolor`, run a function called `setStyles()`that grabs the data items using `Storage.getItem()`and use those values to update page styles.
- If it doesn't, run a function called `populateStorage()`, which uses`Storage.setItem()`to set the item values, then run `setStyles()`.

```
if (!localStorage.getItem("bgcolor")) {
  populateStorage();
}
setStyles();

const populateStorage = () => {
  localStorage.setItem("bgcolor", document.getElementById("bgcolor").value);
  localStorage.setItem("font", document.getElementById("font").value);
  localStorage.setItem("image", document.getElementById("image").value);
};

const setStyles = () => {
  var currentColor = localStorage.getItem("bgcolor");
  var currentFont = localStorage.getItem("font");
  var currentImage = localStorage.getItem("image");

  document.getElementById("bgcolor").value = currentColor;
  document.getElementById("font").value = currentFont;
  document.getElementById("image").value = currentImage;

  htmlElem.style.backgroundColor = "#" + currentColor;
  pElem.style.fontFamily = currentFont;
  imgElem.setAttribute("src", currentImage);
};
```

When would we use the Web Storage API?

Since web storage can store more data than cookies, it's ideal for storing multiple key-value pairs. Like with cookies, this data can be saved only as a string. With localStorage, the data is stored locally on a user's machine, meaning that it can only be accessed client-side. This differs from cookies which can be read both server-side and client-side.

There are a few common use cases for Web storage. One is storing information about a shopping cart and the products in a user's cart. Another is saving input data on forms. You could also use Web storage to store information about the user, such as their preferences or their buying habits.

While we would normally use a cookie to store a user’s ID or a session ID after login, we could use localStorage to store extra information about the user.

You can view what’s in local or session storage by using the Developer Tools. On**Google Chrome**, see the **Application tab**, and on **Firefox**, see the**Storage tab**.

What we learned:

- What cookies are and when to use them
- Differences between cookies and localStorage
- Use cases for cookies and localStorage