Cookies & Sessions

by Sophia

WHAT'S COVERED

In this lesson, you will learn about browser storage and cookies and how they are used to store information about a site or application visitor. You will learn about the traditional cookie and the two new methods, local and session storage, introduced with HTML5.

Specifically, this lesson will cover the following:

1. Introduction to Web Storage Options

Prior to HTML5, the only storage options we had available to us were databases, file storage on the server, and small text files on the client called **cookies**. The cookies themselves were small text files stored on the client's computer, and they allowed websites and web applications to recognize previous visitors and users to customize their experience.

With the introduction of HTML5, developers are now able to store data and information directly in the browser itself. This new type of **web storage** not only simplifies usage for developers but also provides greater capacity and user privacy over traditional cookies. In the next tutorial section, we will explore the use of traditional cookies and the newer HTML5 web storage.

TERMS TO KNOW

Cookie
A small text file created by a webserver and stored on a client's system to identify the client.

Web Storage
Storage that resides within a client's browser and is available to websites and web applications.

2. Web Cookies

Cookies have been in use since just before the year 2000 and were used to recognize users to retrieve their information, data, and preferences and apply them to a site or web application.

Cookies themselves are simple text files, limited to no more than 4 KB (kilobytes), that are created by the server and stored on the client's browser cache. These files can contain authentication, identification, and preference data, as well as browsing history. Cookies are either session based or persistent. **Session cookies** do not include an Expires date or Max-Age value and are removed from the client's system once the browser tab has been closed. **Persistent cookies** have a value for either Expires or Max-Age and will remain on the client's system until either of the values is reached or if the user clears the browser's cache.

Session Cookie
A cookie that does not have a set expiration time and expires when the user closes the browser window or tab.

Persistent Cookie
A cookie that has a set expiration time and remains on the client's system until that time.

2a. Cookie Privacy Concerns

Cookie files have always been a topic of discussion when it comes to user privacy. This is primarily due to sites that use cookies, particularly **third-party cookies**, which can also track users' browsing history on other websites. This means if you visit Website A, which places a tracking cookie, and then you visit Website B and then C, the owner of the cookie from Website A will likely be able to see your history on Websites B and C. This is partially how banner ads on websites conduct targeted advertising. For example, you just visited Amazon and looked at a particular pair of shoes, and after navigating to a completely different website, you now see an ad for those exact shoes.

Due to concerns over the lack of privacy created by the use of web cookies, U.S. and European lawmakers made it a requirement for websites to allow users to opt in or opt out of having cookies stored on their systems.

Another privacy concern regarding cookies is the fact that each time a request is made to a webserver, and it was that webserver that placed the cookie on the client, the cookie is transmitted to the server and then back to the client. Should the encryption on the site ever fail (often due to an expired secure socket layer (SSL) certificate and administrator oversight), the cookie information could be exposed to the public Internet.

As a result of the privacy concerns and new legislation surrounding cookies, as well as the development of HTML5, there has been a shift toward using a newer type of web storage. Let us take a brief look at how to implement cookies before moving on to the newer options for browser storage.

TERM TO KNOW

Third-Party Cookie
A cookie created by an organization other than the one that owns the website that the user is visiting.

2b. Using Cookies

Cookies are set, read, updated, and deleted using JavaScript and the document.cookie object from the DOM. Each cookie contains a key–value pair, an expiration date, and a path within the site to which the cookie belongs.

To set a cookie, we simply need to assign a new string to the document.cookie object. The object will then take the string and parse it out into an actual cookie in the browser's cache.

EXAMPLE Creating a new cookie

```
<script>
    document.cookie = "username=John Doe; expires=Thu, 18 Dec 2023 12:00:00 UTC; path=/";
</script>
```

In the above example, we see that a cookie for the "username" key was set to "John Doe," along with its expiration time and date and the path to the root of the site. The same cookie can be updated by simply reassigning the string to document.cookie with the same key name but a new value.

EXAMPLE Updating an existing cookie

```
<script>
  document.cookie = "username=Jane Doe; expires=Thu, 18 Dec 2023 12:00:00 UTC; path=/";
</script>
```

We can create new cookies the same way but give the key a new name.

EXAMPLE Creating an additional new cookie

```
<script>
  document.cookie = "pref_ViewMode=dark; expires=Thu, 18 Dec 2023 12:00:00 UTC; path=/";
</script>
```

We set a new cookie for "pref_ViewMode" to track the user's preferred mode (light mode vs. dark mode).

If we need to remove or delete a cookie, our only option is to update the cookie's expiration date to a time and date that has already passed. We do not even need to specify the key's value, just the key and the passed date.

EXAMPLE Removing a cookie

```
<script>
  document.cookie = "username=; expires=Thu, 18 Dec 2020 12:00:00 UTC; path=/";
</script>
```

To retrieve a cookie, we need to retrieve the cookie string from document.cookie and then separate the parts of the cookie. Then, we can locate and return the value. We will use a JavaScript function that will accept the key name of the cookie, get and separate the whole cookie string, and then use a loop to step through each key–value pair until we find the match. Then, we return the value or an empty string if no match is found.

EXAMPLE Creating a new cookie

```
<script>
    function getCookie(cname)
    {
let name = cname + "=";
let decodedCookie = decodeURIComponent(document.cookie);
let carray = decodedCookie.split(';');
for (let x = 0; x < carray.length; x++)
{
    let cookie = carray[x];
    while (cookie.charAt(0) == ' ')
              cookie = cookie. Substring(1);
    if (cookie.indexOf( name ) == 0)
    return cookie. Substring(name.length, cookie.length);
}
return "";
    }
</script>
```

STEP BY STEP

The above function operates using the following logic:

1. The function accepts the cookie's key name (cname) as an argument.
2. Concatenate the "=" character on the end of the cname; store it in name.
3. Use decodeURIComponent (document.cookie) to retrieve the string of cookies.
4. Then, split the string of cookies (decodedCookies) using the semicolon as the delimiter in the .split(";") function and store each cookie in an array (carray).
5. Then, start a `for` loop, which steps through each cookie.
    a. Get 1 cookie from carray[x].
    b. Check if the first character is a space; if so, do the following:
        i. Reduce the cookie by removing the first character.
        ii. Recheck if the first character is a space.

c. Check if the value of (name) is a match with the first portion of the cookie; if so, do this:
    i. If the cookie is found, return just the cookie value by returning the substring starting at the end of (name), which would be the "=" character, and ending at the last character, excluding the very last character, which is the semicolon.

6. If nothing was found in the loop, return an empty string.

WATCH

View the following video for more on cookies.

3. Browser Storage

The next advancement in storing data on the client's system emerged with HTML5, improving traditional cookies by making them easier to use, increasing the amount of data they can hold, and having them only reside on the client unless explicitly sent to the server by the developer's code. Browser storage using HTML5 uses key–value pairs, just like a cookie.

The new browser storage comes in two forms: session and local. **Session storage** is active on the browser for as long as the browser tab or window is open and can accommodate up to 5 MB (megabytes). **Local storage** has no expiration and can accommodate up to 10 MB. Both options only support string data in plain text.

Another difference is that session storage is only accessible from the current tab, while local storage is accessible from any tab in the browser window.

DID YOU KNOW

It is very important to note that there is nothing safe or secure about using browser storage! Browser storage should only be used to store nonsensitive data and information, as it is vulnerable to cross-site script (XSS) attacks.

TERMS TO KNOW

Session Storage
Data that is created and stored on the client's system by JavaScript and only remains on the system until the browser window or tab is closed.

Local Storage
Data that is created and stored on the client's system by JavaScript and remains on the system indefinitely until the client removes it.

3a. Using Session Storage

JavaScript contains the necessary objects and methods needed to create, retrieve, update, and remove items from the browser storage. Regarding session storage, we use the sessionStorage object and its internal

methods to handle browser storage values. This is useful for storing information about the client's current visit to the site, and it would no longer be needed once the user closes the browser.

EXAMPLE Creating sessionStorage key–value pairs

```
<script>
sessionStorage.setItem("fname", "John");
sessionStorage.setItem("lname", "Doe");

let userName = sessionStorage.getItem("fname") + " " + sessionStorage.getItem("lname");

let welcomeMsg = document.createElement("h2");
welcomeMsg.innerHTML = "Welcome " + userName + ", nice to have you back!";
document.getElementById("header").appendChild(welcomeMsg);
</script>
```

The example above used the sessionStorage object's setItem() method to create and store two key–value pairs. We then use .getItem() to retrieve the values from the browser storage and use them to create an h2 welcome message that includes the user's name.

The data in the browser storage can be updated by calling the setItem() method, specifying the existing key to be updated, and providing a new value.

EXAMPLE Updating the session storage value

```
<script>
sessionStorage.setItem( "fname", "John" );
sessionStorage.setItem( "fname", "Jane" );
alert( "Welcome, " + sessionStorage.getItem("fname") + ", nice to see you!" );
</script>
```

EXAMPLE The rendered output is as follows:

> Welcome, Jane, nice to see you!

We can also remove session storage using two methods. The first method allows us to remove a specific data value from session storage, while the second will remove all session data. By calling the sessionStorage.removeItem() and providing the key name as the only argument, the data will be removed from the session storage. To clear all session data, call the sessionStorage.clear() method.

EXAMPLE Deleting a session storage value

```
<script>
```

```
sessionStorage.removeItem("fname");
</script>
```

EXAMPLE Deleting all session storage values

```
<script>
sessionStorage.clear();
</script>
```

BIG IDEA

Session storage is ideal for user preferences and caching data retrieved from the server to avoid having to request data repetitively. Preferences such as font size, light or dark mode, and so on can be carried from one page to the next using session data. This way, when a new page is loaded into the browser, JavaScript can read the session data and apply the preferences to the page. This prevents the user from having to constantly change to dark mode after navigating to a different page on the same site.

Additionally, session data can be used to store information and data that was specifically requested from the server. For example, an internal web application for managing orders could make the initial request for the current orders from the database and cache the data using session storage. The data can then be used throughout a session and on multiple pages without needing to rerequest the data from the server. This is particularly helpful in minimizing data usage for mobile users.

WATCH

View the following video for more on session web storage.

3b. Using Local Storage

Local storage works very much the same way as session storage and uses the same methods to set, retrieve, and remove key–value pairs. The primary differences are that we access the getItem(), setItem(), removeItems(), and clear() methods through the localStorage object instead of sessionStorage, and the capacity of local storage is double that of session storage (10 MB).

The other difference between sessionStorage and localStorage is that localStorage does not expire and remains on the client's system even after the tab or window is closed. This is ideal for situations where the user might want to access a resource or document offline after it has been downloaded into localStorage. For example, when using a web application to edit a document, as is the case with Microsoft Office 365, where you can create and edit documents using a browser, there are times when you may not be able to access the Internet and need to continue working on the document. In such cases, the document can be loaded into localStorage, and the user can continue to access and edit the document, even when they have no connection to the Internet.

Due to the higher capacity of localStorage and the fact that it does not expire, this is an ideal solution for storing documents and other large resources to make them available offline and without the user having to explicitly download the file to their system.

Similar to the previous example, another scenario for the use of localStorage is in the case of an Internet outage. If your Internet goes out while editing a document online, the site could temporarily store the document in localStorage until the connection to the Internet can be restored. Additionally, data stored in localStorage is accessible from any browser tab or window.

WATCH

View the following video for more on local web storage.

SUMMARY

In this lesson, you learned about the different **web browser storage options** available to web developers for storing and retrieving information about a client on the client's system. You learned about the older method, which uses **cookies**, or small text files, stored on the user's system. **Privacy concerns over the use of cookies** were also discussed. Finally, you learned about the newer options for **browser storage** on the client's system, which include **session storage** for short-term data and **local storage** for persistent data.

Source: This Tutorial has been adapted from "The Missing Link: An Introduction to Web Development and Programming " by Michael Mendez. Access for free at https://open.umn.edu/opentextbooks/textbooks/the-missing-link-an-introduction-to-web-development-and-programming. License: Creative Commons attribution: CC BY-NC-SA.

TERMS TO KNOW

Cookie
A small text file created by a webserver and stored on a client's system to identify the client.

Local Storage
Data that is created and stored on the client's system by JavaScript and remains on the system indefinitely until the client removes it.

Persistent Cookie
A cookie that has a set expiration time and remains on the client's system until that time.

Session Cookie
A cookie that does not have a set expiration time and expires when the user closes the browser window or tab.

Session Storage
Data that is created and stored on the client's system by JavaScript and only remains on the system until the browser window or tab is closed.

Third-Party Cookie

A cookie created by an organization other than the one that owns the website that the user is visiting.

Web Storage

Storage that resides within a client's browser and is available to websites and web applications.