Quiz

Describe the difference between a cookie, `sessionStorage` and `localStorage` in browsers

High Medium Web APIs JavaScript

Edit on GitHub

TL;DR

All of the following are mechanisms of storing data on the client, the user's browser in this case. `localStorage` and `sessionStorage` both implement the Web Storage API interface.

- Cookies: Suitable for server-client communication, small storage capacity, can be persistent or session-based, domain-specific. Sent to the server on every request.
- `localStorage` : Suitable for long-term storage, data persists even after the browser is closed, accessible across all tabs and windows of the same origin, highest storage capacity among the three.
- `sessionStorage` : Suitable for temporary data within a single page session, data is cleared when the tab or window is closed, has a higher storage capacity compared to cookies.

Here's a table summarizing the 3 client storage mechanisms.

Property	Cookie	`localStorage`	`sessionStorage`
Initiator	Client or server. Server can use `Set-Cookie` header	Client	Client
Lifespan	As specified	Until deleted	Until tab is closed

Property	Cookie	`localStorage`	`sessionStorage`
Persistent across browser sessions	If a future expiry date is set	Yes	No
Sent to server with every HTTP request	Yes, sent via `Cookie` header	No	No
Total capacity (per domain)	4kb	5MB	5MB
Access	Across windows/tabs	Across windows/tabs	Same tab
Security	JavaScript cannot access `HttpOnly` cookies	None	None

Storage on the web

Cookies, `localStorage`, and `sessionStorage`, are all storage mechanisms on the client (web browser). It is useful to store data on the client for client-only state like access tokens, themes, personalized layouts, so that users can have a consistent experience on a website across tabs and usage sessions.

These client-side storage mechanisms have the following common properties:

- This means the clients can read and modify the values (except for `HttpOnly` cookies).
- Key-value based storage.
- They are only able to store values as strings. Non-strings will have to be serialized into a string (e.g. `JSON.stringify()`) in order to be stored.

Use cases for each storage mechanism

Since cookies have a relatively low maximum size, it is not advisable to store all your client-side data within cookies. The distinguishing properties about cookies are that cookies are sent to the server on every HTTP request so the low maximum size is a feature that prevents your HTTP requests from being too large due to cookies. Automatic expiry of cookies is a useful feature as well.

With that in mind, the best kind of data to store within cookies is small pieces of data that needs to be transmitted to the server, such as auth tokens, session IDs, analytics tracking IDs, GDPR cookie consent, language preferences that are important for authentication, authorization, and rendering on the server. These values are sometimes sensitive and can benefit from the `HttpOnly`, `Secure`, and `Expires`/`Max-Age` capabilities that cookies provide.

`localStorage` and `sessionStorage` both implement the Web Storage API interface. Web Storages have a generous total capacity of 5MB, so storage size is usually not a concern. The key difference is that values stored in Web Storage are not automatically sent along HTTP requests.

While you can manually include values from Web Storage when making AJAX/`fetch()` requests, the browser does not include them in the initial request / first load of the page. Hence Web Storage should not be used to store data that is relied on by the server for the initial rendering of the page if server-side rendering is being used (typically authentication/authorization-related information). `localStorage` is most suitable for user preferences data that do not expire, like themes and layouts (if it is not important for the server to render the final layout). `sessionStorage` is most suitable for temporary data that only needs to be accessible within the current browsing session, such as form data (useful to preserve data during accidental reloads).

The following sections dive deeper into each client storage mechanism.

Cookies

Cookies are used to store small pieces of data on the client side that can be sent back to the server with every HTTP request.

- **Storage capacity**: Limited to around 4KB for all cookies.
- **Lifespan**: Cookies can have a specific expiration date set using the `Expires` or `Max-Age` attributes. If no expiration date is set, the cookie is deleted when the browser is closed (session

cookie).

- **Access**: Cookies are domain-specific and can be shared across different pages and subdomains within the same domain.
- **Security**: Cookies can be marked as `HttpOnly` to prevent access from JavaScript, reducing the risk of XSS attacks. They can also be secured with the `Secure` flag to ensure they are sent only when HTTPS is used.

```
// Set a cookie for the name/key `auth_token` with an expiry.
document.cookie =
  'auth_token=abc123def; expires=Fri, 31 Dec 2024 23:59:59 GMT; path=/';

// Read all cookies. There's no way to read specific cookies using `document.cook
// You have to parse the string yourself.
console.log(document.cookie); // auth_token=abc123def

// Delete the cookie with the name/key `auth_token` by setting an
// expiry date in the past. The value doesn't matter.
document.cookie = 'auth_token=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/';
```

It is a pain to read/write to cookies. `document.cookie` returns a single string containing all the key/value pairs delimited by `;` and you have to parse the string yourself. The `js-cookie` npm library provides a simple and lightweight API for reading/writing cookies in JavaScript.

A modern native way of accessing cookies is via the Cookie Store API which is only available on HTTPS pages.

```
// Set a cookie. More options are available too.
cookieStore.set('auth_token', 'abc123def');

// Async method to access a single cookie and do something with it.
cookieStore.get('auth_token').then(...);

// Async method to get all cookies.
cookieStore.getAll().then(...);

// Async method to delete a single cookie.
```

```
cookieStore.delete('auth_token').then(() =>
  console.log('Cookie deleted')
);
```

The CookieStore API is relatively new and may not be supported in all browsers (supported in latest Chrome and Edge as of June 2024). Refer to caniuse.com for the latest compatibility.

`localStorage`

`localStorage` is used for storing data that persists even after the browser is closed and reopened. It is designed for long-term storage of data.

- **Storage capacity**: Typically around 5MB per origin (varies by browser).
- **Lifespan**: Data in `localStorage` persists until explicitly deleted by the user or the application.
- **Access**: Data is accessible within all tabs and windows of the same origin.
- **Security**: All JavaScript on the page have access to values within `localStorage`.

```
// Set a value in localStorage.
localStorage.setItem('key', 'value');

// Get a value from localStorage.
console.log(localStorage.getItem('key'));

// Remove a value from localStorage.
localStorage.removeItem('key');

// Clear all data in localStorage.
localStorage.clear();
```

`sessionStorage`

`sessionStorage` is used to store data for the duration of the page session. It is designed for temporary storage of data.

- **Storage Capacity**: Typically around 5MB per origin (varies by browser).

- **Lifespan**: Data in `sessionStorage` is cleared when the page session ends (i.e., when the browser or tab is closed). Reloading the page does not destroy data within `sessionStorage`.
- **Access**: Data is only accessible within the current tab or window. Different tabs or windows with the same page will have different `sessionStorage` objects.
- **Security**: All JavaScript on the same page have access to values within `sessionStorage` for that page.

Amazon 52/61

Mark complete

```js
sessionStorage.setItem('key', 'value');

// Get a value from sessionStorage.
console.log(sessionStorage.getItem('key'));

// Remove a value from sessionStorage.
sessionStorage.removeItem('key');

// Clear all data in sessionStorage.
sessionStorage.clear();
```

Notes

There are also other client-side storage mechanisms like IndexedDB which is more powerful than the above-mentioned technologies but more complicated to use.

References

- What is the difference between localStorage, sessionStorage, session and cookies?

Report an issue

Edit on GitHub

SWAG overflow
front end engineer
[fruhnt end en-juh-neer]
noun
One who writes code that looks flawless in Chrome, chaotic in Safari, and doesn't load in Internet Explorer.

Sponsor: SwagOverflow

The Component Library Procrastinator Tee — Why reuse when you can rebuild?

A true front end engineer never reuses, they only reinvent. This sleek black tee sums up the reality of constantly rebuilding component libraries instead of using the one from last month. Clean, sharp, and painfully accurate, it's the perfect fit for devs who can't resist a fresh start.

Get yours now ->