**Stop Using localStorage**

[[Julien Etienne](https://medium.com/@julienetienne?source=post_page-----64a6d6805da8--------------------------------)](https://medium.com/@julienetienne?source=post_page-----64a6d6805da8--------------------------------)

[Julien Etienne](https://medium.com/@julienetienne?source=post_page-----64a6d6805da8--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F272fb8983fbd&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40julienetienne%2Fstop-using-localstorage-64a6d6805da8&user=Julien+Etienne&userId=272fb8983fbd&source=post_page-272fb8983fbd----64a6d6805da8---------------------post_header-----------)

5 min read

·

Dec 22, 2023

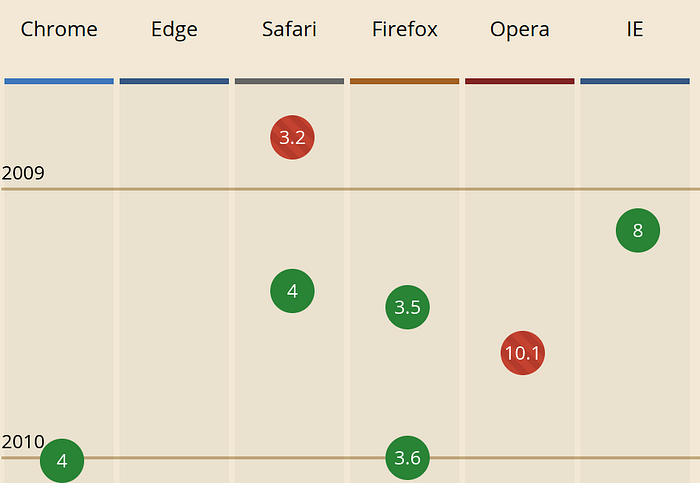
2.2K

34



This is a real-life representation of localStorage

[**localStorage**](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage) is OLD

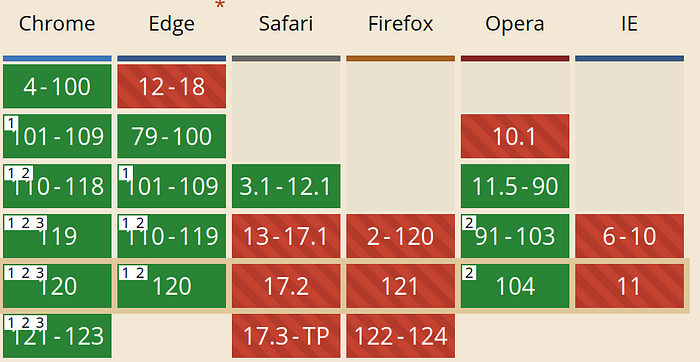


Not crazy old, but it came out around [2009](https://caniuse.com/?search=localstorage) and it was a primitive design from the jump.

Let’s cut to the chase with some bullet points since our attention spans are in disarray these days:

* **One data type**: You can only store strings. If you want to store anything else you have to serialize it to a string and deserialize to retrieve.
* **Historically slow**: localStorage is somewhat slow at storing and retrieving data, which makes it undesirable for applications that require frequent transactions.
* **Limited storage**: localStorage has a [5MB cap](https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria)
* **Serialization gotchas**: If you are not familiar with the gotchas when storing data in localStorage you may be creating bugs in your application. These type-errors are not always obvious, especially for new developers.
* **Blocking operations**: localStorage is not asynchronous and will block your application. It may even make your animations choppy in the right conditions. Asynchronicity is fundamental to creating fluid applications, especially for mobile devices

**What about WebSQL?**



[WebSQL](https://developer.chrome.com/blog/deprecating-web-sql) aimed to be a simple SQL database interface for the web. It had some decent support but eventually faced challenges that led to depreciation.

**Why did they drop the baby?**

* **Single-Vendor Implementation**: WebSQL was primarily a webkit thing (Initially Chrome and Safari). A lack of support from other major browser vendors (Mozilla and Microsoft) made developer adoption close to non-existent in the commercial world.
* **No W3C Standardization:** this is crucial for adoption. [W3 appeared to drop the proposal in 2010](https://www.w3.org/TR/webdatabase/).
* **Competition with IndexedDB**: IndexedDB was gaining more traction and support during the rise of WebSQL. Unlike WebSQL, IndexedDB was designed to be a standard, cross-browser solution.
* **Security Concerns**: Several developers and security specialists showed concerns regarding WebSQL’s safety. They were sceptical of various aspects including lack of permission controls and SOL style vulnerabilities.

Eventually, IndexedDB became the “recommended” standard for client-side storage, being seen as more robust and cross-browser friendly. But what does “recommended” really mean when the majority of experienced front-end developers *at the time of this article* have avoided it like the plague?

It’s also worth stating that despite its shortcomings, at the time WebSQL was highly praised amongst the web community and was a worthy competitor.

**What about Cookies?**

Cookies were created in 1994 by Lou Montulli, a web browser programmer at Netscape Communications.

Some of you were not even born when cookies were effing stuff up. The title of this article should be “Stop Using Cookies and localStorage” but that’s a very difficult fight, (and yes we should use secure cookies)

* **Size Limitations**: Cookies are typically limited to roughly 4KB per domain.
* **Data Sent with Every Request**: Cookies are sent with every HTTP request to the associated domain. If your data doesn’t need to be transmitted with every request, this can result in unnecessary overhead and increased bandwidth usage.
* **Security Concerns**: Cookies are more susceptible to XSS. Because cookies are automatically included with every request to the domain, they can be targeted by malicious scripts.
* **Expiration and Lifetime**: Cookies were designed to expire by a given date.
* **Increased latency**: Because cookies are automatically included in every HTTP request to the domain, they typically lead to increased latency thus making your website slower to load.

**Why IndexedDB**



* **Better Performance**: Unlike localStorage, IndexedDB operates asynchronously, preventing blocking operations. *(The API is event-driven not Promise based)*
* **Ample Storage Quota**: IndexedDB provides a larger storage quota *(dependent on the browser, OS and available storage)* compared to [localStorage’s 5MB cap](https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria" \t "_blank).
* **Reliability and Structured Data**: Storing and retrieving data in localStroage can yield unpredictable results if not done properly. IndexedDB reduces common type-coercion and embraces the [structuredClone algorithm](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm" \t "_blank), ensuring data integrity.

**You probably don’t want to use IndexedDB directly**

It’s absolutely not fun, and you have better things to do than to fight with a hostile storage just to be disappointed by how it’s not what you anticipated.

The reason you want to use a library is because mostly they:

* Are promise based
* Are easier to use
* Reduce boilerplate code
* Focus on more meaningful things.

I don’t recommend using libraries bigger than e.g. 10kB gzipped. All these kilobytes add up and those 50kB+ libs are not doing anything meaningful for your real-world scenarios.

The one problem I found with most IndexedDB libraries is how they are mostly oriented around versioning, which you probably don’t need at all, especially if you’re just looking for a reasonable localStorage alternative.

*Quick plug: I made*[*db64*](https://github.com/julienetie/db64)*, a library to focus on just the more relative aspects of IndexedDB.*

If you do need versioning or cursors I’d recommend [idb](https://github.com/jakearchibald/idb" \t "_blank), which is a comprehensive library that covers all niche cases well.

**Conclusion**

IMO **nobody should be using localStorage in this day and age**. New developers will have a better experience playing with [Promise()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) or [async/ await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await) than trying to figure out why the number ‘0’ is making their conditions *truthy*.

As a recap there’s a speed advantage, it’s non-blocking, you can store types reliably and you can even use cursors to iterate over entries, if you’re into that sort of thing you can easily build a client-side search-engine with accumulated data fetches that don’t make your animations fidget like how localStorage blocks and interferes with them 👀.

For persistent storage, IndexedDB is the better tool for the job especially if you’re using a wrapper library to make life easier.

Thanks for the read :)