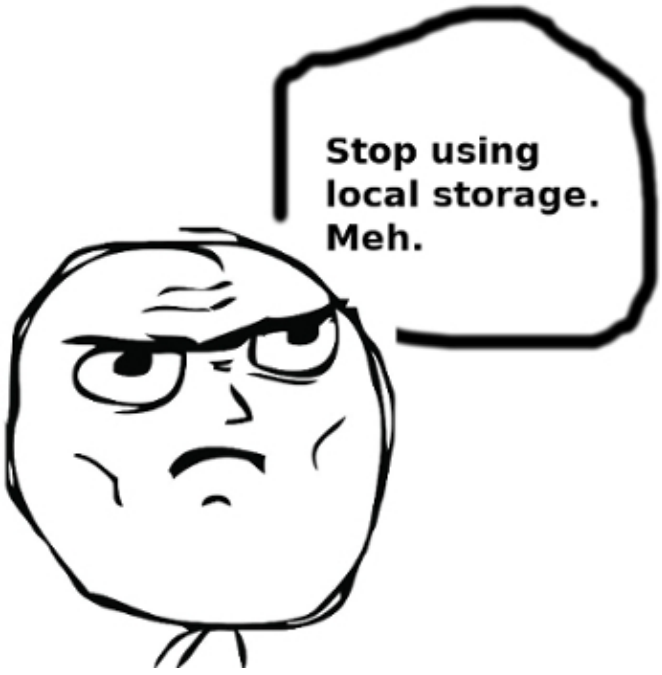
RANDALL DEGGES (/)

Please Stop Using Local Storage



Seriously. Just stop it already.

I don't know what it is, exactly, that drives so many developers to store session information in local storage (https://developer.mozilla.org/en-US/docs/Web/API/Storage/LocalStorage), but whatever the reason: the practice needs to die out. Things are getting completely out of hand.

Almost every day I stumble across a new website storing sensitive user information in local storage and it bothers me to know that so many developers are opening themselves up to catastrophic security issues by doing so.

Let's have a heart-to-heart and talk about local storage and why you should stop using it to store session data.

What is Local Storage?

I'm sorry if I was a bit grumpy earlier. You don't deserve that! Heck, you might not even be familiar with what local storage is, let alone be using it to store your session information!

Let's start with the basics: local storage is a new feature of HTML5 that basically allows you (a web developer) to store any information you want in your user's browser using JavaScript. Simple, right?

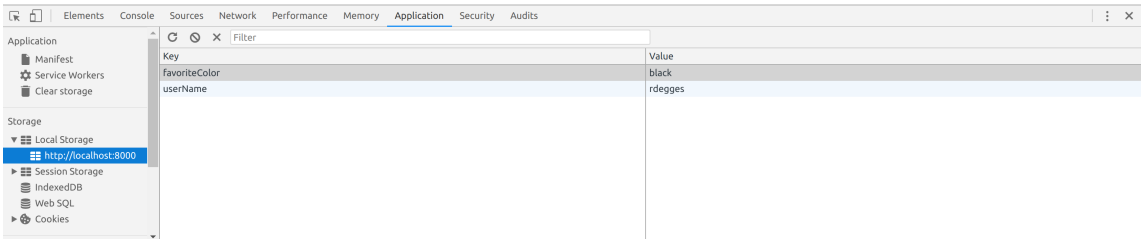
In practice, local storage is just one big old JavaScript object that you can attach data to (or remove data from). Here's an example of some JavaScript code that stores some of my personal info in local storage, echoes it back to me, and then (optionally) removes it:

```
// You can store data in local storage using either syntax
localStorage.userName = "rdegges";
localStorage.setItem("favoriteColor", "black");

// Once data is in localStorage, it'll stay there forever until it is
// explicitly removed
alert(localStorage.userName + " really likes the color " + localStorage.favori
teColor + ".");

// Removing data from local storage is also pretty easy. Uncomment the lines
// below to destroy the entries
//localStorage.removeItem("userName");
//localStorage.removeItem("favoriteColor");
```

If you run the JavaScript code above in your browser on a test HTML page, you'll see the phrase "rdegges really likes the color black." in an alert message. If you then open up your developer tools, you'll be able to see the that both the `userName` and `favoriteColor` variables are both stored in local storage in your browser:



Now you might be wondering if there's some way to use local storage so that the data you store is automatically deleted at some point and you don't need to manually delete every single variable you put in there. Luckily, the HTML5 working group (shout out!) has your back. They added something called sessionStorage to HTML5 which works *exactly* the same as local storage except that all data it stores is automatically deleted when the user closes their browser tab.

What's Cool About Local Storage?

Now that we're on the same page about what local storage is, let's talk about what makes it cool! Even though the whole point of this article is to dissuade you from using local storage to store session data, local storage still has some interesting properties.

For one thing: it's pure JavaScript! One of the annoying things about cookies (the only real alternative to local storage) is that they need to be created by a web server. Boo! Web servers are boring and complex and hard to work with.

If you're building a static site (like a single page app, for instance), using something like local storage means your web pages can run independently of any web server. They don't need any backend language or logic to store data in the browser: they can just do it as they please.

This is a pretty powerful concept and one of the main reasons that local storage is such a hit with developers.

Another neat thing about local storage is that it doesn't have as many size constraints as cookies. Local storage provides at least 5MB of data storage across all major web browsers, which is a heck of a lot more than the 4KB (maximum size) that you can store in a cookie.

This makes local storage particularly useful if you want to cache some application data in the browser for later usage. Since 4KB (the cookie max size) isn't a lot, local storage is one of your only real alternative

options.

What Sucks About Local Storage

OK. We talked about the good, now let's spend a minute (or two!) talking about the bad.

Local storage is soooo *basic*. WHEW. I feel better already getting that off my chest. Local storage is just an incredibly basic, simple API.

I feel like most developers don't realize just how basic local storage actually is:

- It can only store string data. Boo. This makes it pretty useless for storing data that's even slightly more complex than a simple string. And sure, you *could* serialize everything including data types into local storage, but that's an ugly hack.
- It is synchronous. This means each local storage operation you run will be one-at-a-time. For complex applications this is a big no-no as it'll slow down your app's runtime.
- It can't be used by web workers (https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers) =/ This means that if you want to build an application that takes

advantage of background processing for performance, chrome extensions, things like that: you can't use local storage at all since it isn't available to the web workers.

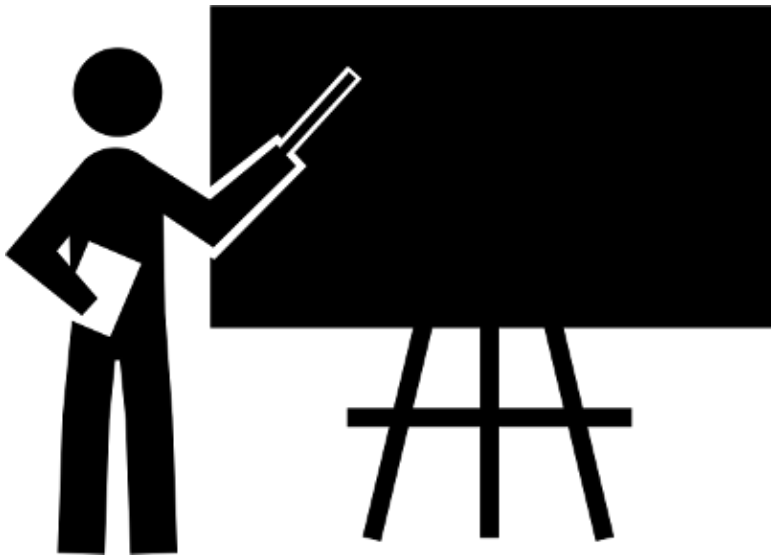
- It still limits the size of data you can store (~5MB across all major browsers). This is a fairly low limit for people building apps that are data intensive or need to function offline.

- Any JavaScript code on your page can access local storage: it has no data protection whatsoever. **This is the big one for security reasons** (as well as my number one pet peeve in recent years).

To keep it short, here's the only situation in which you should use local storage: when you need to store some publicly available information that is not at all sensitive, doesn't need to be used in a high performance app, isn't larger than 5MB, and consists of purely string data.

If the app you're using doesn't fit the above description: *don't use local storage*. Use something else (more on this later).

Why Local Storage is Insecure and You Shouldn't Use it to Store Sensitive Data



Here's the deal: most of the bad things about local storage aren't all that important. You can still get away with using it but you'll just have a slightly slower app and minor developer annoyance. But security is different. The security model of local storage IS really important to know and understand, since it will dramatically affect your website in ways you may not realize.

And the thing about local storage is that it is *not secure*! Not at all! Everyone who uses local storage to store sensitive information such as session data, user details, credit card info (even temporarily!) and anything else you wouldn't want publicly posted to Facebook is doing it wrong.

Local storage *wasn't designed* to be used as a secure storage mechanism in a browser. It was designed to be a simple string only key/value store that developers could use to build slightly more complex single page apps. That's it.

What's the most dangerous thing in the entire world? That's right! JavaScript.

Think about it like this: when you store sensitive information in local storage, you're essentially using the most dangerous thing in the world to store your most sensitive information in the worst vault ever created: not the best idea.

What the problem really boils down to is cross-site scripting attacks (XSS (https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))). I won't bore you with a full explanation of XSS, but here's the high level:

If an attacker can run JavaScript on your website, they can retrieve all the data you've stored in local storage and send it off to their own domain. This means anything sensitive you've got in local storage (like a user's session data) can be compromised.

Now, you might be thinking "So what? My website is secure. No attacker can run JavaScript on my website."

And that's a reasonable point. If your website is *truly* secure and no attacker can run JavaScript code on your website then you are technically safe, but in reality that is incredibly hard to achieve. Let me explain.

If your website contains *any* third party JavaScript code included from a source outside your domain:

- Links to bootstrap
- Links to jQuery
- Links to Vue, React, Angular, etc.
- Links to any ad network code
- Links to Google Analytics
- Links to any tracking code

Then you are currently at risk for having an attacker run JavaScript on your website. Let's say your website has the following script tag embedded inside it:

```
<script src="https://awesomejslibrary.com/minified.js"></script>
```

In this case, if awesomejslibrary.com is compromised and their `minified.js` script gets altered to:

- Loop through all data in local storage
- Send it to an API built to collect stolen information

… then you are completely screwed. In this situation the attacker would have easily been able to compromise anything you had stored in local storage and you would never notice. Not ideal.

As engineers, I think we're frequently susceptible to thinking that we would never embed third party JavaScript in our websites. But in the real world, this scenario rarely plays out.

At most companies the marketing team directly manages the public website using different WYSIWYG editors and tooling. Can you *really* be sure that nowhere on your site are you using third party JavaScript? I'd

argue “no”.

So to err on the side of caution and dramatically reduce your risk for a security incident: **don’t store anything sensitive in local storage**.

PSA: Don’t Store JSON Web Tokens in Local Storage

While I feel like I made myself clear that you should never *ever* store sensitive information in local storage in the previous section, I feel the need to specifically call out JSON Web Tokens (JWTs).

The biggest security offenders I see today are those of us who store JWTs (session data) in local storage. Many people don’t realize that JWTs are essentially the same thing as a username/password.

If an attacker can get a copy of your JWT (https://stackoverflow.com/questions/34259248/what-if-jwt-is-stolen), they can make requests to the website on your behalf and you will never know. Treat your JWTs like you would a credit card number or password: don’t ever store them in local storage.

There are thousands of tutorials, YouTube videos, and even programming classes at universities and coding bootcamps incorrectly teaching new developers to store JWTs in local storage as an authentication mechanism. **THIS INFORMATION IS WRONG.** If you see someone telling you to do this, run away!

What to Use Instead of Local Storage

So with all of local storage's shortcomings, what should you use instead? Let's explore the alternatives!

Sensitive Data

If you need to store sensitive data, you should always use a server-side session. Sensitive data includes:

- User IDs
- Session IDs
- JWTs

- Personal information
- Credit card information
- API keys
- And anything else you wouldn't want to publicly share on Facebook

If you need to store sensitive data, here's how to do it:

- When a user logs into your website, create a session identifier for them and store it in a cryptographically signed cookie. If you're using a web framework, look up "how to create a user session using cookies" and follow that guide.

- Make sure that whatever cookie library your web framework uses is setting the `httpOnly` cookie flag. This flag makes it impossible for a browser to read any cookies, which is *required* in order to safely use server-side sessions with cookies. Read Jeff Atwood's article (https://blog.codinghorror.com/protecting-your-cookies-httponly/) for more information. He's the *man*.

- Make sure that your cookie library also sets the `SameSite=strict` cookie flag (to prevent CSRF (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)) attacks), as well as the `secure=true` flag (to ensure cookies can only be set over an encrypted connection).

- Each time a user makes a request to your site, use their session ID (extracted from the cookie they send to you) to retrieve their account details from either a database or a cache (depending on how large your website is)

- Once you have the user's account info pulled up and verified, feel free to pull any associated sensitive data along with it

This pattern is simple, straightforward, and most importantly: *secure*. And yes, you can most definitely scale up a large website using this pattern. Don't tell me that JWTs are "stateless" and "fast" and you have to use local storage to store them: you're wrong!

Non-String Data

If you need to store data in the browser that isn't sensitive and isn't purely string data, the best option for you is IndexedDB. It's an API that lets you work with a database-esque object store in the browser.

What's great about IndexedDB is that you can use it to store typed information: integers, floats, etc. You can also define primary keys, handle indexing, and create transactions to prevent data integrity issues.

A great tutorial for learning about (and using) IndexedDB is this Google tutorial (https://developers.google.com/web/ilt/pwa/working-with-indexeddb).

Offline Data

If you need your app to run offline, your best option is to use a combination of IndexedDB (above) along with the Cache API (which is a part of Service Workers).

The Cache API allows you to cache network resources that your app needs to load.

A great tutorial for learning about (and using) the Cache API is this Google tutorial (https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api).

Please Stop Using Local Storage

Now that we've had a chance to talk about local storage, I hope you understand why you (probably) shouldn't be using it.

Unless you need to store publicly available information that:

- Is not at all sensitive
- Doesn't need to be used in an ultra high performance app
- Isn't larger than 5MB
- Consists of purely string data

… **don't use local storage!** Use the right tool for the job.

And please, please, whatever you do, do not store session information (like JSON Web Tokens) in local storage. This is a very bad idea and will open you up to an extremely wide array of attacks that could absolutely cripple your users.

Have a question? Shoot me an email (mailto:r@rdegges.com).

Stay safe out there =)

NOTE: For those of you who made it this far who are wondering why I didn't specifically call out Content Security Policy (https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP) as a way to mitigate the effects of XSS, I specifically chose not to include this

because it cannot help in the situation I described above. Even if you use CSP to whitelist all third party JavaScript domains, that does nothing to prevent XSS if the third party provider is compromised.

And while we're at it: subresource integrity (https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity) (while cool) is also not a global solution to this issue. For most marketing tools, ad networks, etc. (which are by far the most commonly used types of third party JavaScript), subresource integrity is almost never used as the providers of those scripts *want* to change them frequently so they can silently update functionality for their users.

UPDATE: I'm not the only one who thinks you should never store anything sensitive in local storage. So does OWASP (https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet#Local_Storage)

> … In other words, any authentication your application requires can be bypassed by a user with local privileges to the machine on which the data is stored. Therefore, it's recommended not to store any sensitive information in local storage.

PS: If you read this far, you might want to follow me on twitter (https://twitter.com/rdegges) or github (https://github.com/rdegges) and subscribe via RSS (/feed.xml) or email below (*I'll email you new articles when I publish them*).

Email Address

SUBSCRIBE