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