SEE FURTHER DOWN FOR MDN SECTIONS  
  
UDEMY Discussion: https://www.udemy.com/course/nodejs-the-complete-guide/learn/lecture/12097918#questions/8663708  
  
<https://stackoverflow.com/questions/54258233/do-i-have-to-store-tokens-in-cookies-or-localstorage-or-session>  
  
**This answer is based on the stateless approach and therefore it doesn't talk about the traditional session management**

You have asked two altogether different questions:

1. Shopping cart - which is more related to business functionality
2. OAuth 2 & JWT - which is related to security and authentication

As an user of an ecommerce website, I'd expect that any item I add to my shopping cart from my mobile device while commuting to my workplace, should be available in the cart when I login to the website from my PC after reaching home. Therefore, the cart data should be saved in the back-end DB and linked to my user account.

When it comes to authentication using OAuth 2.0, the JWT access token and / or refresh token need to be stored somewhere in the client device, so that once the user authenticates himself by providing login credentials, he doesn't need to provide his credentials again to navigate through the website. In this context, the browser local storage, session storage and cookies are all valid options. However, note that here the cookie is not linked to any session on the server side. In other words, the cookie doesn't store any session id. The cookie is merely used as a storage for access token which is passed to the server with every http request and the server then validates the token using the digital signature to ensure that it is not tampered and it is not expired.

Although all three storage options for access and / or refresh tokens are popular, cookie seems to be the most secured option when used in the correct way.

To understand this better, I recommend you read [this](https://stackoverflow.com/a/54011649/1235935) and [this](https://stackoverflow.com/a/53988717/1235935) along with the OAuth 2.0 specification.

## Update On 16-Feb-2019

I said earlier that cookie seems to be the most secured options. I'd like to further clarify the point here.

The reason I think browser localStorage and sessionStorage do not provide enough security for storing auth tokens are as follows:

1. If XSS occurs, the malicious script can easily read the tokens from there and send them to a remote server. There on-wards the remote server or attacker would have no problem in impersonating the victim user.
2. localStorage and sessionStorage are not shared across sub-domains. So, if we have two SPA running on different sub-domains, we won't get the SSO functionality because the token stored by one app won't be available to the other app within the organization. There are some solutions using iframe, but those look more like workarounds rather than a good solution. And when the response header X-Frame-Options is used to avoid clickjacking attacks with iframe, any solution with iframe is out of question.

These risks can, however, be mitigated by using a fingerprint (as mentioned in [OWASP JWT Cheat Sheet](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.md#token-sidejacking)) which again in turn requires a cookie.

The idea of fingerprint is, generate a cryptographically strong random string of bytes. The Base64 string of the raw string will then be stored in a HttpOnly, Secure, SameSite cookie with name prefix \_\_Secure-. Proper values for Domain and Path attributes should be used as per business requirement. A SHA256 hash of the string will also be passed in a claim of JWT. Thus even if an XSS attack sends the JWT access token to an attacker controlled remote server, it cannot send the original string in cookie and as a result the server can reject the request based on the absence of the cookie. The cookie being HttpOnly cannot be read by XSS scripts.

Therefore, even when we use localStorage and sessionStorage, we have to use a cookie to make it secured. On top of that, we add the sub-domain restriction as mentioned above.

Now, the only concern about using a cookie to store JWT is, CSRF attack. Since we use SameSite cookie, CSRF is mitigated because cross-site requests (AJAX or just through hyperlinks) are not possible. If the site is used in any old browser or some other not so popular browsers that do not support SameSite cookie, we can still mitigate CSRF by additionally using a CSRF cookie with a cryptographically strong random value such that every AJAX request reads the cookie value and add the cookie value in a custom HTTP header (except GET and HEAD requests which are not supposed to do any state modifications). Since CSRF cannot read anything due to same origin policy and it is based on exploiting the unsafe HTTP methods like POST, PUT and DELETE, this CSRF cookie will mitigate the CSRF risk. This approach of using CSRF cookie is used by all modern SPA frameworks. The Angular approach is mentioned [here](https://angular.io/guide/http#security-xsrf-protection).

Also, since the cookie is httpOnly and Secured, XSS script cannot read it. Thus XSS is also mitigated.

It may be also worth mentioning that XSS and script injection can be further mitigated by using appropriate content-security-policy response header

\*\*\*\* Not that I disagree with the general answer, but if one succeeds in doing an XSS penetration, then they don't need the token  
  
  
\*\*\*\*\*\*@Kaiido Thanks for the feedback. But with this approach the XSS has to do whatever harm it intends to do from the user's browser only. Remote impersonation is far less likely. – [Saptarshi Basu](https://stackoverflow.com/users/1235935/saptarshi-basu" \o "3,457 reputation)  
  
  
**SECTION 1: Confidentiality, Integrity, and Availability**

Confidentiality: Only those authorized and relevant stakeholders can access sensitive data

Integrity: Deals with the authenticity of the information. IE: Protecting admin routes to protect data, protecting against CSRF spoofing the site