Link: [Robert Kawecki Sessions vs tokens: a how-to guide for implementing authentication state in a product](https://www.youtube.com/watch?v=ZljWXMnMluk)

* A web app authentication system consists of the following: User, Credentials, Authentication Logic and Secret. The secret is provided to not require secrets to be sent multiple times.
* With HTTP Basic Auth (HTTP0), the **Credentials** (username + password) is sent in the header with every request. Logout is not possible. This method shouldn’t be used as it has no Secret and the Credentials are unencrypted.
* When we use Session, then we have a Secret (Session ID). This Secret **should only be stored in an HTTPOnly Cookie** and not in LocalStorage in Frontend or SessionStorage as these other places can be accessed.
* The Session ID should be unique, random and as non-deterministic (hard to guess) and as high entropy as possible. It has 5 parts and is a UID and doesn’t tell us anything.
* Session IDs are typically stored in a database and resolved by looking it up there.
* Since Session IDs are opaque, so they can’t be used directly for authorization. This is because it isn’t possible to distinguish one Session ID from another and tell which one should offer which privilege.
* In Session-based authentication, the Session ID is stored in a database that correlates with the user’s info such as (user\_id, role, etc.). It is then resolved to user\_id and role.
* By contrast, a JWT already contains all the user info and doesn’t need to be resolved against a database for user\_id and role.

**Pros & Cons of Sessions**

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Instant Invalidation – User’s privieleges can be revoked by changing details in session database. | Latency – Resolving user details from Session ID using Session Database is time consuming. |
| Listability – It is possible to list the users who accessed the system using their session details. | Coupling to Central DB – The Session Database is centralized. Unsuitable for microservices. Difficult to allow other technologies to interface with the Session DB (Eg – Java, C++, etc.) |
| Auxiliary Data – It is possible to store some extra data (like shopping cart data) in the Session DB. | Race Condition – Session middleware works on RMWL (Read, Modify Write, Loop). Thus, it is possible for race conditions to occur. {Sessions load at the beginning of our request handler, then modifications are allowed to it and then the session is written back, replacing the previous session state.} |

**Tips to Make Sessions Faster:**

* Ensure that sessions don’t store unnecessary data (images, stylesheets, fav icon, etc.).
* Use a fast In-memory database like Redis or shard your database.
* To prevent race conditions store a reference to the data in the session, instead of directly storing the data there. (Eg – Store a reference to a Shopping Cart in the session and then use a **transaction** to modify that).

**Pros and Cons of JWT**

**“Fixing” JWT**:

* Store the shopping cart as a reference in local storage.
* Tokens should have proper expiry times (not too long or too short).
* Access Token + Refresh Token.

**Hybrid:** Use Access + Refresh Tokens and also build a database where the issued refresh tokens are stored. This way**, 1 refresh token = 1 session**. Revoke the **refresh token** in the database if there is any unauthorized use, this will still allow intruders to continue using their present access token, but only until it expires. After access token expiry, they won’t be able to get a new refresh token again.

When building a JWT, we should make the business logic independent of the session / token mechanism. The token should have field to identify user and his role.

**Rules:**

* Sessions are fine for most user-facing apps. They should be used if it is your first time building an app and also if you don’t have any good reason for using JWT.
* Use JWT for bulk APIs and pre-signed URLs (Access Token only. Eg – An API that takes 5 minutes to do a bulk import.) (If the Refresh Tokens are going to be issued rarely).
* (Access Token + Refresh Token) should be used for federation and microservices.
* The secret-passing logic should be away from the business logic.
* Know how to invalidate the access-token / refresh-token and how long it will take.

**Some 3rd Party Solutions:**

* Some 3rd Party solutions for authentication are: Keycloak (Open Source), auth0 (Commercial) and SuperTokens (Oper Source / Commercial). All of these use hybrid (JWT + Session) method of authentication.
* Read SuperTokens’ libraries. They offer a lot of insight into how to build a hybrid Session + JWT system.

Some Other Tips:

* We are responsible for authorization even if we don’t manage authentication.
* We should pick the right tool for the job.
* Talk to cybersec / incident response folks.

**Links**:

* **Evert Pot – JWT is a bad default.**
* **Express-session – basic session middleware**
* **Jwt.io – jwt token debugger.**
* **Keycloak – Open Source federated login.**
* **Auth0**
* **SuperTokens**