[**REST API security Stored token vs JWT vs OAuth**](https://softwareengineering.stackexchange.com/questions/298973/rest-api-security-stored-token-vs-jwt-vs-oauth)

Let me tell, how I understand all this stuff. If I understand something incorrectly, please let me know.

As far as REST API is stateless as well as WEB in general, we need to send some auth data in each request (cookies, token....). I know three widely used mechanisms to authenticate user

1. Token with HTTPS. I have used this approach a lot of times it is good enough with HTTPS. If user provides correct password and login, he will receive token in response, and will use it for the further requests. Token is generated by the server and stored, for instance in the table separate or the same where user info is stored. So for each request server checks if user has token and it is the same as in the database. Everything is pretty straightforward.
2. JWT Token. This token is self-descriptive, it contains all necessary information about the token itself, user cannot change for example expiration date or any other claim, because this token is generated (signed) by the server with secret keyword. This is also clear. But one big problem, personally for me, how to invalidate token.
3. OAuth 2. I don't understand why this approach should be used when communication is established directly between server and client. As far as I understand, OAuth server is used to issue token with restricted scope to allow other applications access user information without storing password and login. This is great solution for the social networks, when user wants to sign up on some page, server can request permissions to get user info, for instance from twitter or Facebook, and fill registration fields with user data and so on.

Consider mobile client for online store.

First question should I prefer JWT over first type token? As far as I need login/logout user on mobile client, I need to store somewhere token or in case of JWT, token should be invalidated on logout. Different approaches are used to invalidate token one of this is to create invalid token list (black list). Hmm. Table/file will have much bigger size than if token was stored in table and associated with user, and just removed on logout.

So what are benefits of JWT token?

Second question about OAuth, should I use it in case of direct communication with my server? What is the purpose of one more layer between client and server only to issue token, but communication will be not with oauth server but with the main server. As I understand OAuth server is responsible only for giving third-party apps permissions (tokens) to access user private information. But my mobile client application is not third-party.

## **Session IDs/ Cookies (Spring Rest security)**

**Pros:**

* Easy to code both the client and server.
* Easy to destroy a session when someone logs out.

**Cons:**

* The server side periodically needs to delete expired sessions where the client didn't logout.
* Every HTTP request requires a look up to the data store.
* Storage requirements grow as more users have active sessions.
* If there are multiple front end HTTP servers the stored session data needs to be accessible by all of them. This could be a bit more work than storing it on one server. The bigger issues are the data store becomes a single point of failure and it can become a bottleneck.

## **JSON Web Tokens (JWT)**

In the second case the data is stored in a JWT that is passed around instead of on the server.

**Pros:**

* The server side storage issues are gone.
* The client side code is easy.

**Cons:**

* The JWT size could be larger than a session ID. It could affect network performance since it is included with each HTTP request.
* The data stored in the JWT is readable by the client. This may be an issue.
* The server side needs code to generate, validate, and read JWTs. It's not hard but there is a bit of a learning curve and security depends on it.

Anyone who gets a copy of the signing key can create JWTs. You might not know when this happens.

There was (is?) a bug in some libraries that accepted any JWT signed with the "none" algorithm so anyone could create JWTs that the server would trust.

* In order to revoke a JWT before it expires you need to use a revocation list. This gets you back to the server side storage issues you were trying to avoid.

## **OAuth**

Often OAuth is used for authentication (i.e. identity) but it can be used to share other data like a list of content the user has purchased and is entitled to download. It can also be used to grant access to write to data stored by the third party. You might use OAuth to authenticate users and then use server side storage or JWT for the session data.

**Pros:**

* No code for users to sign up or reset their password.
* No code to send an email with a validation link and then validate the address.
* Users do not need to learn/write-down another username and password.

**Cons:**

* You depend on the third party in order for your users to use your service. If their service goes down or they discontinue it then you need to figure something else out. Eg: how do you migrate the user's account data if their identity changes from "foo@a.com" to "bar@b.com"?
* Usually you have to write code for each provider. eg Google, Facebook, Twitter.
* You or your users might have privacy concerns. The providers know which of their users use your service.
* You are trusting the provider. It is possible for a provider to issue tokens that are valid for one user to someone else. This could be for lawful purposes or not.

### **Miscellaneous**

* Both session IDs and JWTs can be copied and used by multiple users. You can store the client IP address in a JWT and validate it but that prevents clients from roaming from say Wi-Fi to cellular.