# Advanced React & Patterns

## JWTs and Authentication:

Usually, we used to store authentication information for a user’s session in the backend servers. But since SPAs came we needed some way to keep it on the frontend as well as we do a lot of stuff on the client side and don’t want to trigger APIs on every few clicks. And hence came the JWTs or (JSON Web Tokens).

**JWT** (JSON Web Token) is a **compact**, **self-contained** way to represent **claims between two parties** — typically a **client and a server**.

**A JWT looks like this:**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJ1c2VySWQiOiIxMjM0IiwidXNlcm5hbWUiOiJqb2huZG9lIn0.

SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c  
  
**It has 3 parts:**

1. **Header**: metadata (e.g. algorithm)
2. **Payload**: the data (e.g. user ID)
3. **Signature**: ensures token integrity (signed with a secret)

## Why JWT?

JWTs are used to **verify user identity** without needing to **store session info on the server** — they're **stateless**, which is great for scalability.

Instead of the server keeping session data in memory, the token itself **carries the user data**, and the server can **verify the signature** to trust it.

**✅ JWT in a Typical Authentication Flow**

**🔁 Two tokens:**

1. **Access Token**
   * Short-lived (e.g. 5–15 minutes)
   * Used to access protected routes
2. **Refresh Token**
   * Long-lived (e.g. days or weeks)
   * Used to **get a new access token** when the old one expires

**🔄 Flow Overview:**

1. **User logs in** (e.g. /api/login):
   * Server verifies credentials
   * Returns: { accessToken, refreshToken }
2. **Frontend stores the tokens**:
   * Access Token → usually in **memory** or **Authorization header**
   * Refresh Token → in **httpOnly cookie** (recommended) or localStorage (less secure)
3. **User makes requests**:
   * Include Authorization: Bearer <accessToken> header
4. **If access token expires**:
   * Use refresh token to call /api/refresh-token to get a new one
   * Update access token in memory
5. **Logout**:
   * Frontend deletes tokens
   * Optionally call server to invalidate refresh token

**HTTP Only cookies**You don’t touch these from the frontend if you want to store something in httpOnly cookie the backend must do this with the response it should send a set cookie and that sets the value of refresh token in the browser.  
  
Now, how do you use it?  
You don’t basically in the API call you just need to set  
  
credentials: ‘include’   
  
and this’ll automatically attach your refresh token to you refresh-access-token call

**Why not store access token in httpOnly cookie as well?**

There are basically 3 reasons:  
i) Makes the API more prone to CSRF attacks

Since it’s now stored in the browser other websites can trigger your API which is known as Cross Site Resource Forgery and for this you might have to again introduce more security measures on the backend which will increase the complexity there.

ii) You lose the frontend granularity

Many of the SDKs, tools and libraries need you to pass the access token. Few examples of them are Apollo GraphQL, Stripe and many more

iii) Short-lived

If you store the access token in httpOnly cookie you won’t know when it’s going to expire and eventually your api calls will start failing with 401 unauthorized and only at that time you’ll know that your access token has expired which might make the page laggy, unresponsive or might log a user out without them knowing. But if you get it in the response and store it in the memory you can decode it read the expiration time for it and refresh the access token before it expires.