Interview questions - **Authentication & Authorization**

✅ 1. Why is JWT commonly used for stateless authentication in distributed systems?

Answer:
JWT (JSON Web Token) is commonly used in stateless authentication because:

- **Self-contained**: A JWT contains all necessary claims (user identity, roles, permissions, expiration) within the token itself. This removes the need for server-side session storage.
- **Stateless scalability**: Since the token can be validated without querying a central session store, it's ideal for distributed systems and microservices.
- **Easy to transmit**: JWTs are compact, URL-safe, and can be easily sent in HTTP headers (usually as a `Bearer` token).
- **Standardized**: JWTs follow a standard format (header, payload, signature), and are widely supported across languages and platforms.
- **Security**: JWTs can be signed using HMAC or RSA, ensuring authenticity and integrity.

However, care must be taken with:

- Token expiration and renewal
- Secure token storage (e.g., avoid storing in localStorage in browsers)
- Revocation strategies, since tokens are not easily invalidated once issued

---

✅ 2. How does OAuth2 differ from OpenID Connect?

Answer:

- **OAuth2** is an **authorization framework** that allows a third-party application to obtain limited access to a user's resources without exposing credentials.

    - Example: Letting a third-party app post on your behalf on Twitter.

- **OpenID Connect (OIDC)** is an **authentication layer** built on top of OAuth2.
    - It allows clients to verify the user's identity and obtain basic profile info.
    - It introduces the **ID token** (typically a JWT), which carries user identity information.

Summary:

Feature	OAuth2	OpenID Connect
Purpose	Authorization	Authentication + Authorization
Token type	Access token	Access token + ID token
User identity info	Not included	Included in ID token
Standard login flows	No	Yes

✅ 3. Explain the difference between RBAC and ABAC. Which one is more suitable for a highly dynamic environment?

Answer:

- **RBAC (Role-Based Access Control)**:
    - Access is granted based on user roles (e.g., Admin, Editor, Viewer).
    - Policies are tied to roles, and users are assigned one or more roles.
- **ABAC (Attribute-Based Access Control)**:
    - Access is granted based on a combination of attributes (user, resource, environment).
    - Example attributes: `department = HR`, `project = X`, `access_time = business_hours`.

RBAC is simpler to manage but less flexible.
ABAC provides **fine-grained and context-aware** access control, making it better suited for **highly dynamic environments** like multi-tenant cloud platforms or systems with complex policies.

✅ 4. What are the advantages of using Single Sign-On (SSO) in a system?

Answer:
Single Sign-On (SSO) allows users to authenticate once and gain access to multiple applications or systems without re-entering credentials.

Advantages:

- **Improved User Experience**: Users log in once to access all connected systems.
- **Reduced Credential Fatigue**: Less need to remember/manage multiple passwords.
- **Stronger Security**: Centralized authentication allows enforcement of better security policies (e.g., MFA, password rotation).
- **Reduced Helpdesk Load**: Fewer password-related support tickets.
- **Easier Compliance & Auditing**: Central logs and policy enforcement simplify security audits.

SSO is often used in enterprises, SaaS products, and B2B platforms, typically via protocols like **SAML**, **OAuth2**, or **OpenID Connect**.

✅ 5. What are some potential security concerns with token-based authentication?

Answer:
While token-based auth (like JWT) is powerful, there are several concerns to be aware of:

- **Token Theft**: If stored insecurely (e.g., localStorage), tokens can be stolen via XSS attacks.
- **Lack of Revocation**: JWTs are usually valid until expiry; revoking them mid-life is non-trivial without maintaining a token blacklist.
- **Token Replay**: If intercepted (MITM attack), a token can be reused.
- **Long-lived Tokens**: Extended expiry increases risk if token is compromised.
- **Signature Verification Issues**: If secret keys or algorithms are misconfigured (e.g., using `none` algorithm), tokens can be forged.

Mitigations:

- Use HTTPS everywhere
- Store tokens securely (e.g., HttpOnly cookies)
- Implement short-lived tokens with refresh tokens
- Validate token signature, audience, issuer, expiration
- Monitor for suspicious activity and support forced logout