Stateful vs Stateless Authentication

The server keeps track of the client's session (server stores the state).

How it works:

1. User logs in → server verifies credentials.
2. Server creates a **session** (a record in memory or DB).
   Example session:
3. {
4. "sessionId": "xyz123",
5. "userId": "101",
6. "role": "admin"

}

The server sends a **session ID** to the client in a cookie.

On each request, client sends cookie → server checks session store → validates user.

✅ **Advantages**

- Easy to implement.
- Server can invalidate sessions (force logout).
- Useful for apps where you need live session control.

❌ **Disadvantages**

- Server must store sessions (more memory usage).
- Hard to scale (each server must share session data).
- Requires sticky sessions or distributed cache (Redis).

✅ **Advantages**

- Scalable (no server memory for sessions).
- Works across multiple servers easily.
- Faster requests (no DB/session lookup each time).

❌ **Disadvantages**

- Token revocation is tricky (server can't "delete" a token once issued).
- If the token is stolen, the attacker has access until it expires.
- Typically needs refresh tokens for long sessions.

🔷 2. Stateless Authentication

👉 The server does NOT store session state. Instead, the client carries all the necessary info (usually in JWT).

How it works:

1. User logs in → server verifies credentials.
2. Server creates a **JWT (JSON Web Token)**, signs it, and sends it to the client.
3. Client stores token (localStorage, cookie, memory).
4. On each request, the client sends a token in headers (Authorization: Bearer <token>).
5. Server verifies JWT signature (using secret/public key). No session lookup needed.

🔑 What is a Token?

A **token** is like a **digital key** (a string of characters) that proves a user's identity after login.

- Instead of storing login credentials (like email & password) with every request, the server gives the user a token after successful authentication.
- The client (browser/app) uses this token in every request to prove "Hey, I am already logged in".

🔷 How Tokens Work in Authentication

1. **Login Request:**
   User sends credentials (email + password) to the server.
2. **Verification:**
   The server checks credentials against the database.
3. **Token Generation:**
   If valid, the server generates a token (e.g., **JWT – JSON Web Token**) and sends it back to the client.
4. **Storage:**
   - The client stores the token (in **localStorage**, **sessionStorage**, or **cookies**).
5. **Accessing Protected Routes:**
   - On subsequent requests, the client includes the token (usually in the **Authorization header**) →
     Authorization: Bearer <token>
6. **Server Validation:**
   - The server validates the token before giving access to protected resources.

◈ Types of Tokens

1. **Session Tokens (Stateful):**
    - Stored on the server.
    - The server keeps track of the user session.
    - Example: Classic **session IDs with cookies**.
2. **JWT (Stateless Tokens):**
    - Encoded JSON that contains user data.
    - Self-contained → no need to store session on server.
    - Example:
    - {
    - "header": { "alg": "HS256", "typ": "JWT" },
    - "payload": { "userId": 123, "role": "admin" },
    - "signature": "securehash"
    - }

◈ Advantages of Tokens

- **Stateless (for JWT):** No need to store the session in the database.
- **Scalable:** Useful in distributed systems (like microservices).
- **Secure:** Credentials are exchanged only once (at login).

◈ Drawbacks of Tokens

- If JWT is stolen → attacker gets access until it expires.
- Cannot be invalidated easily (unless you maintain a blacklist).

✔ In short:

- Tokens replace storing user credentials in every request.
- **Stateful tokens (session IDs)** need server-side storage.
- **Stateless tokens (JWT)** are self-contained and don't require server memory for sessions.

🍪 What are Cookies?

A **cookie** is a **small piece of data** (text file) that a server sends to the browser.
The browser stores it and sends it back to the server with each request.

👉 Think of it as a **note** that the server gives to your browser, saying:
"Remember this info and show it to me every time you come back."

🔹 Key Features of Cookies

- Stored in the browser (client-side).
- Sent automatically to the server with every HTTP request (if they match the domain & path).
- Can store small data like **session IDs, preferences, tokens, and authentication data**.
- Have properties like:
    - **Name/Value pair** → sessionId=abc123
    - **Expiration** → When the cookie should be deleted.
    - **Secure** → Only sent over HTTPS.
    - **HttpOnly** → Cannot be accessed by JavaScript (for security).
    - **SameSite** → Controls cross-site behavior (helps prevent CSRF).

🔹 Cookies vs Tokens

- **Cookie:** Storage mechanism (where data lives).
- **Token:** Authentication credential (what proves identity).
  👉 Together: A **token can be stored inside a cookie** for authentication.