**1️⃣ What is JWT?**

**JWT (JSON Web Token)** is an **open standard** (RFC 7519) for securely transmitting information between two parties as a **JSON object**.

* It is **digitally signed**, so the receiver can verify it hasn’t been tampered with.
* It can be signed using:
  + **HMAC** (secret key)
  + **RSA/ECDSA** (public/private keys)

JWT is mostly used for **stateless authentication** in modern web & microservices.

**2️⃣ Why JWT is popular**

* **Stateless** — no server session storage needed (token contains all the data).
* **Lightweight** — works well with mobile, SPA, and microservices.
* **Cross-domain** — easy to share between services via HTTP headers.
* **Performance** — avoids database lookups for each request.

**3️⃣ JWT Structure**

A JWT is made up of **three parts** separated by dots (.):

**1. Header**

* Specifies the token type and signing algorithm.

{

"alg": "HS256",

"typ": "JWT"

}

Encoded with **Base64Url**.

**2. Payload**

* Contains **claims** (information about the user or token).
* Types of claims:
  + **Registered** (standard): sub (subject), iat (issued at), exp (expiration), iss (issuer)
  + **Public**: custom claims shared across apps
  + **Private**: app-specific claims

{

"sub": "avinash",

"role": "ADMIN",

"iat": 1723111111,

"exp": 1723114711

}

**3. Signature**

* Verifies the token hasn’t been changed.
* Created by:

HMACSHA256(

base64UrlEncode(header) + "." + base64UrlEncode(payload),

secretKey

)

Example decoded JWT:

Header:

{

"alg": "HS256",

"typ": "JWT"

}

Payload:

{

"sub": "avinash",

"role": "ADMIN",

"iat": 1723111111,

"exp": 1723114711

}

Signature:

(Signed with secret key)

**4️⃣ How JWT Works in Authentication**

**Step 1: Login**

1. Client sends username/password to **Auth API**.
2. Server authenticates credentials.
3. Server generates JWT with user details & expiry.
4. Server sends JWT to client.

**Step 2: Access Resource**

1. Client sends a request to API with:

Authorization: Bearer <jwt-token>

1. API Gateway or Resource Server:
   * Extracts JWT from header.
   * Verifies signature with the secret/public key.
   * Checks expiration.
   * Reads claims for authorization.
2. If valid → process request.  
   If invalid/expired → return 401 Unauthorized.

**5️⃣ JWT in Spring Security**

When using Spring Security, JWT replaces the **session-based** mechanism.

**Core components:**

* **JwtTokenProvider** → generates and validates tokens.
* **AuthenticationManager** → authenticates credentials during login.
* **OncePerRequestFilter** → checks JWT in every request.
* **SecurityConfig** → defines URL security rules.

**Spring Boot typical flow:**

1. POST /login → AuthenticationManager verifies credentials → returns JWT.
2. JWTAuthenticationFilter intercepts all requests → extracts & validates token → sets SecurityContext.
3. Controllers read Principal or claims to authorize.

**6️⃣ Advantages of JWT**

✅ Stateless → scalable for microservices.  
✅ Secure if used with HTTPS.  
✅ No session storage.  
✅ Works across different domains.

**7️⃣ Security Best Practices**

* Always use **HTTPS** to avoid token interception.
* Keep JWTs **short-lived** (exp claim).
* Use **refresh tokens** for renewing access tokens.
* Store JWTs securely (HTTP-only cookies or secure local storage).
* Rotate signing keys periodically.
* Avoid storing sensitive info (like passwords) inside JWT payload — it’s base64 encoded, **not encrypted**.

**8️⃣ JWT in Microservices**

In a microservice architecture:

* **API Gateway** validates JWT before forwarding requests to downstream services.
* Internal microservices trust the gateway → they don’t need to re-validate.
* JWT claims can store **user roles**, **permissions**, and **tenant IDs**.

Example:

Client → API Gateway (JWT validation) → User Service / Order Service / Payment Service

If you want, I can next create a **diagram + complete Spring Security + JWT example** where:

* /auth/login generates token
* /secure only accessible with valid token
* Works with microservices & API Gateway