Who are you? Vs What can you do?

AUTHENTICATION & AUTHORIZATION

GATEKEEPER (SIGNATURE CHECKER)

- **Rule Sheet 1: Gatekeeper (Signature Checker)**
- **Your Role:**
  You are the **Gatekeeper**. You only check the **physical sign** on the slip.
- **Rules**
- Look **only at the sign** on the bottom of the slip
- Sign Code is – **ABC345**
- If the sign is **valid** → allow the person inside
- If the sign is **missing or fake** → deny entry
- Do **NOT** read the name or employee ID
- Do **NOT** ask questions

ACCESS CONTROLLER (ID CHECKER)

- **Your Role:**
  You control **who can switch off the light**.
- **Rules**
- Check the **Employee ID written on the slip**
- Only **TWO specific Employee ID** - 5679 & 8903 are allowed
- If ID matches → allow light switch access
- If ID does not match → deny access
- Only if the person is inside, rules apply

AUTHENTICATION (WHO ARE YOU?)

- Verifies **identity**
- Happens first
- Uses credentials (username/password, OTP, token)
- Example: Logging into an app

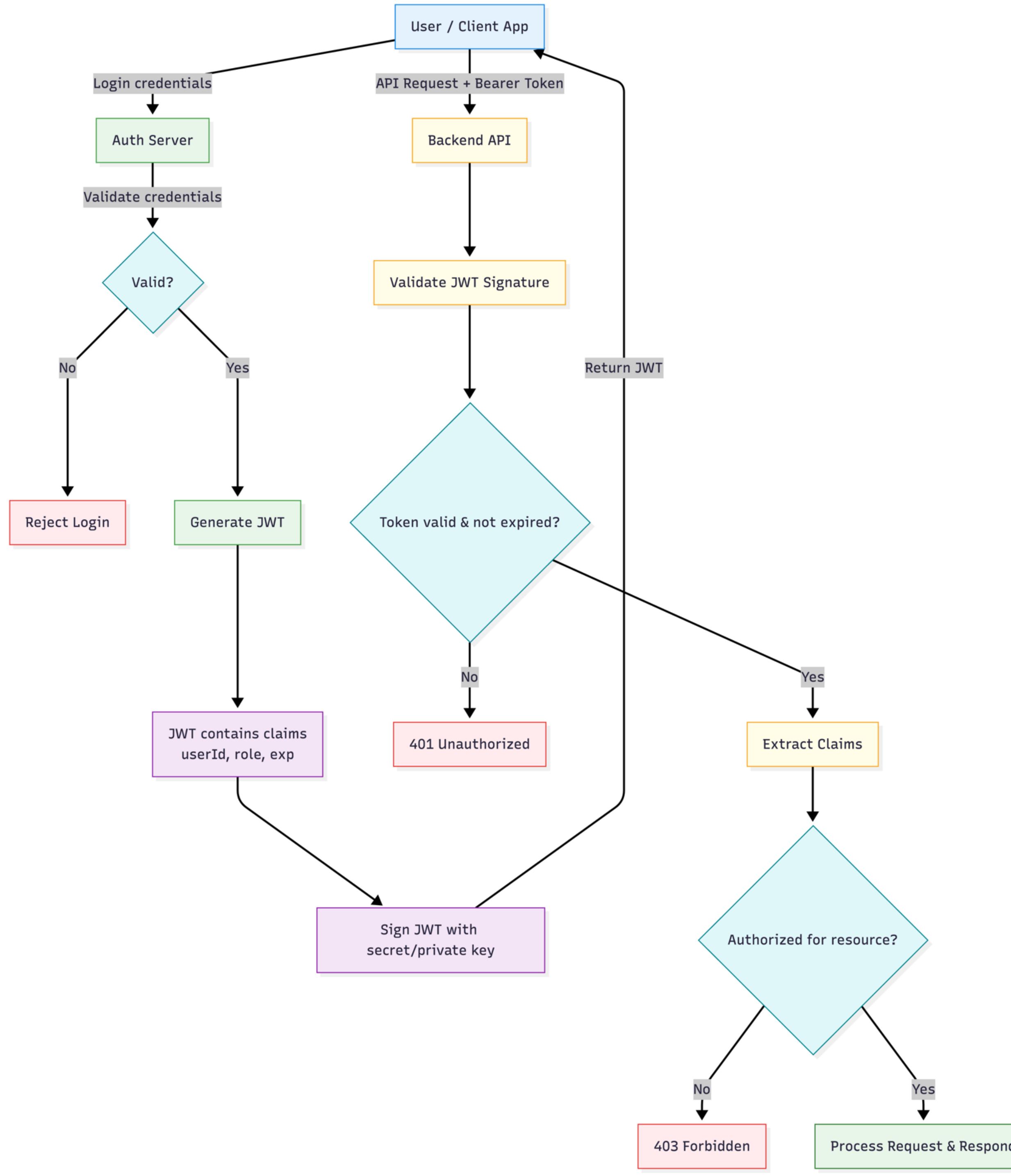
AUTHORIZATION (WHAT CAN YOU DO?)

- Verifies **permissions**
- Happens after authentication
- Uses roles, scopes, policies
- Example: Can you access /admin?

ACCESS TOKEN

- A short-lived token used to **access APIs**
- Usually a **JWT**
- Sent with every request
- **How it's used**
- Client sends token in header
  Authorization: Bearer <access_token>
- Backend validates:
  - Signature
  - Expiry
  - Permissions
- **Why short-lived**
  - Limits damage if stolen
  - Forces re-authentication or refresh

TOKE VALIDATION FLOW



JWT – WHAT IT IS & WHY IT'S USED

- **What is JWT?**
  - JWT (JSON Web Token) is a **compact, self-contained token**
  - Used to **securely transmit claims** between client and server
  - Stateless → server doesn't store session data
- **Why use JWT?**
  - **Stateless auth** → perfect for microservices & APIs
  - Scales well (no session store, no DB hit per request)
  - Works cleanly with **REST, mobile apps, SPAs**
  - Easy to pass via HTTP headers (Authorization: Bearer <token>)

JWT

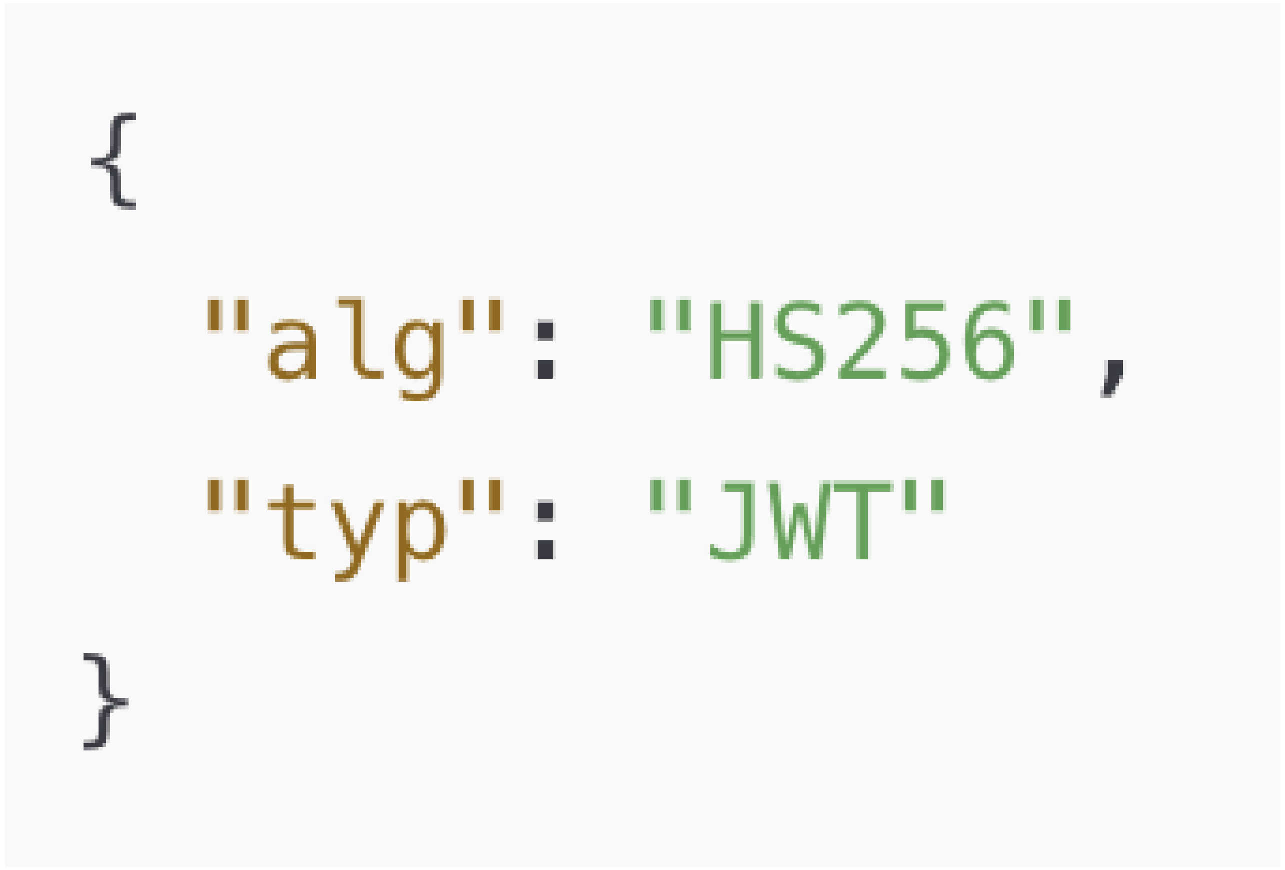
- **Where it fits**
  - After login → server issues JWT
  - Client sends JWT on every request
  - Server validates token → grants access
- **Blunt reality**
  - JWT is great for **auth**, not magic
  - Long-lived JWTs + bad revocation = security risk

JWT STRUCTURE

- **Title:** JWT (JSON Web Token) Structure
- **JWT = 3 parts (dot separated)**

```
Header.Payload.Signature
```

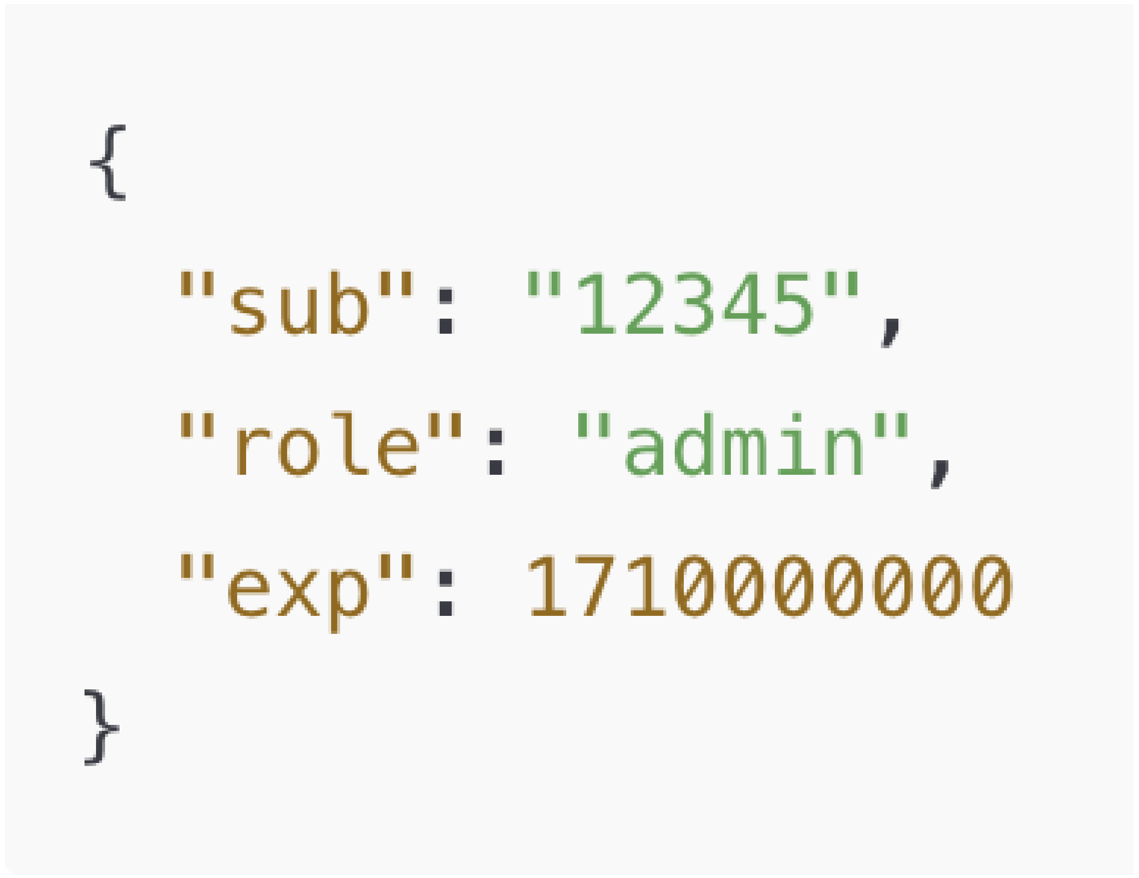
JWT HEADER



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- **What it contains**
- Token type → JWT
- Signing algorithm → HS256 / RS256
- **Truth**
  - Header tells **how** the token is signed
  - Weak algorithm choice = weak security

JWT PAYLOAD



```
{
  "sub": "12345",
  "role": "admin",
  "exp": 1710000000
}
```

- **What it contains**
- User identity → sub, userId
- Authorization data → role, scope
- Token lifetime → iat, exp
- **Hard rule**
  - Payload is **not encrypted**
  - Never put passwords or secrets here

JWT SIGNATURE

```
HMAC(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret / private key
)
```

- **What it does**
- Ensures **integrity**
- Prevents tampering
- **Reality**
- If signature fails → token is trash
- Private key leakage = total compromise

PASSWORD HASHING CONCEPT

- Passwords are **never stored as plain text**
- They are converted into a **one-way hash**
- **How it works**
    - User enters password
    - System hashes it (bcrypt / argon2)
    - Hash is stored, not the password
- **During login**
    - Entered password → hashed again
    - Hashes compared
    - Match = correct password
- **Hard truth**
    - Hashing ≠ encryption (no reverse)
    - If you can “decrypt” it, you did it wrong

WHAT IS OAUTH 2.0

- **OAuth 2.0** is a **delegated authorization framework**.
- OAuth2 lets an app access your data **without knowing your password**.
- **Why OAuth2 exists**
  - Sharing passwords is unsafe
  - Apps need limited, controlled access
  - Users should be able to **revoke access anytime**

REAL-LIFE EXAMPLE – OAUTH 2.0

- “Login with Google”
- You **don’t** give Gmail password to the app
- Google gives the app a **token**
- Token has **limited permissions**
- You can revoke it later
- **What OAuth2 actually does**
  - Issues **Access Tokens**
  - Defines **who can access what**
  - Does **NOT** define how login UI works

OAUTH 2.0 VS JWT

Aspect	**OAuth2**	**JWT**
Type	Framework	Token format
Solves	Authorization	Identity & claims
Token	Issues tokens	Is the token
Can exist alone	❌	✅
Common combo	OAuth2 + JWT	JWT inside OAuth2

SAML

- **What it is:** Authentication + authorization protocol
- **Data format:** XML (heavy)
- **Transport:** Browser redirects + XML assertions
- **Best for:** Enterprise SSO (old-school)
- **User experience:** Slower, clunky
- **Common today:** Mostly legacy

OUTH 2.0

- **What it is:** Authorization framework
- **Data format:** JSON
- **Transport:** REST / HTTP
- **Best for:** APIs, mobile apps, SPAs
- **User experience:** Fast, modern
- **Common today:** ✅ Yes