
WEEK 1 – DAY 6

SESSION vs TOKEN AUTHENTICATION

(Very Detailed, Easy-to-Explain, Interview + Practical Ready)

---

1. Why Authentication State Management Matters

After a user logs in, the system must remember:

"This user is authenticated"

This is called **authentication state management**.

Two common ways:

1. **Session-based authentication**
2. **Token-based authentication (JWT)**

---

2. Cookies (Foundation Concept)

What Is a Cookie?

A **cookie** is a small piece of data stored in the **browser**, sent automatically with every request to the server.

Example:

session_id=abc123xyz

What Cookies Are Used For

- Session tracking
- Authentication
- Preferences

Security Risks of Cookies

- XSS can steal cookies
- CSRF attacks
- Session hijacking

Secure Cookie Flags (Interview Must-Know)

- **HttpOnly** → JS cannot access cookie
- **Secure** → Sent only over HTTPS
- **SameSite** → Prevents CSRF

---

3. Session-Based Authentication

3.1 What Is a Session?

A **session** is server-side storage that keeps user authentication state.

Flow:

- Browser stores **session ID** (cookie)
- Server stores session data

3.2 Session Authentication Flow (Step-by-Step)

1. User logs in
2. Server creates a session
3. Server stores session data
4. Session ID sent to browser as cookie
5. Browser sends cookie with every request
6. Server verifies session ID

3.3 Session Diagram (Think This Way)

Browser —(session_id)—> Server

<— session data stored on server

3.4 Advantages of Sessions

- Simple to implement
- Easy to invalidate (logout)
- Secure for traditional web apps

3.5 Disadvantages of Sessions

- Server memory usage

- Not ideal for distributed systems
- Scaling becomes harder

Interview Line

“Sessions are stateful and stored on the server, which makes scaling harder.”

---

4. Token-Based Authentication (JWT)

4.1 What Is a Token?

A **token** is a piece of data that proves authentication.

The server does **not store session data**.
Everything needed is inside the token.

4.2 What Is JWT?

JWT (JSON Web Token) is a **self-contained token**.

Structure:

HEADER.PAYLOAD.SIGNATURE

---

5. JWT Structure (VERY IMPORTANT)

5.1 Header

Contains:

- Token type (JWT)
- Signing algorithm

Example:

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

5.2 Payload

Contains **claims** (user data).

Example:

```
{
  "user_id": 101,
  "role": "admin",
  "exp": 1710000000
}
```

⚠ Do NOT store sensitive data here
Payload is **Base64 encoded, not encrypted**.

5.3 Signature

Ensures:

- Token integrity
- Token authenticity

Created using:

- Header
- Payload
- Secret key

Interview Line (Very Strong)

“JWT is signed, not encrypted. Anyone can read it, but no one can modify it without the secret.”

---

6. Access Tokens vs Refresh Tokens

6.1 Access Token

- Short-lived (5–15 minutes)
- Sent with every request
- Used to access APIs

Example:

Authorization: Bearer <access_token>

6.2 Refresh Token

- Long-lived
- Used to generate new access tokens
- Stored securely

6.3 Why Two Tokens Are Used

If access token is stolen:

- It expires quickly
- Damage is limited

Interview Line

“Short-lived access tokens reduce the impact of token leakage.”

---

7. Session vs JWT (Interview Comparison)

Feature	Session	JWT
Stored on server	Yes	No
Stateless	No	Yes
Scalability	Harder	Easier
Logout	Easy	Hard
Used in APIs	Rare	Common

---

8. Security Risks in Token Auth (Must Know)

- Token leakage
- Storing JWT in localStorage
- No expiration
- Weak secret key

---

9. Hands-On: Create JWT Login API in Flask

Below is **clean, interview-ready JWT authentication code**.

9.1 Install Required Packages

```
pip install flask flask-jwt-extended
```

9.2 Basic Flask App Setup

```
from flask import Flask, jsonify, request
from flask_jwt_extended import (
    JWTManager, create_access_token,
    create_refresh_token, jwt_required,
    get_jwt_identity
)

app = Flask(__name__)

app.config["JWT_SECRET_KEY"] = "super-secret-key"
jwt = JWTManager(app)
```

9.3 Mock User Database

```
users = {
    "kavya": {"password": "password123", "role": "admin"},
    "arjun": {"password": "userpass", "role": "user"}
}
```

9.4 Login API (Create Tokens)

```
@app.route("/login", methods=["POST"])
def login():
    data = request.get_json()
    username = data.get("username")
    password = data.get("password")

    user = users.get(username)
```

```
    if not user or user["password"] != password:
        return jsonify({"error": "Invalid credentials"}), 401

    access_token = create_access_token(identity=username)
    refresh_token = create_refresh_token(identity=username)

    return jsonify({
        "access_token": access_token,
        "refresh_token": refresh_token
    })
```

9.5 Protected Route (Access Token Required)

```
@app.route("/dashboard")
@jwt_required()
def dashboard():
    current_user = get_jwt_identity()
    return jsonify({"message": f"Welcome {current_user}"})
```

9.6 Refresh Token Endpoint

```
@app.route("/refresh", methods=["POST"])
@jwt_required(refresh=True)
def refresh():
    current_user = get_jwt_identity()
    new_access_token = create_access_token(identity=current_user)
    return jsonify({"access_token": new_access_token})
```

---

10. How This Looks in Real Applications

- Frontend stores access token (memory)
- Refresh token stored securely
- Access token sent in Authorization header
- Refresh endpoint used silently