JSON Web Token

JWT (JSON Web Token) is a compact, URL-safe token format used to securely transmit information between parties as a JSON object.

It's commonly used for authentication and authorization in web applications.

Access Token

- An Access Token is a short-lived token that a client (like a browser or mobile app) sends with API requests to prove the user's identity.

What does it contain?

- An access token usually includes:
  - sub: user ID
  - exp: expiry time (e.g., 15 minutes after it's issued)
  - iat: issued-at time
  - optional: user role, scopes, etc.
- Lifespan:
  - Short-lived (typically 15 minutes to 1 hour)

Refresh Token:

- A Refresh Token is a long-lived token that is used to obtain a new access token without requiring the user to log in again.
- It should be stored securely – ideally in an HTTP-only, Secure cookie. Never store refresh tokens in JavaScript-accessible storage (like localStorage), to avoid XSS attacks.

Structure of JWT

JWTs have three parts, separated by dots (.):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
eyJ1c2VySWQiOiIxMjM0NSIsIm5hbWUiOiJKb2huIERvZSJ9.
```

```
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

1. Header – typically includes algorithm (alg) and token type (typ)

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

2. Payload – data (claims), like userId, exp (expiry), etc.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "exp": 1516242622
}
```

3. Signature – cryptographic signature to verify integrity

```
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secr
```

Signing Algorithms

HS256 – HMAC + SHA256 (symmetric: same secret for sign & verify)

RS256 – RSA + SHA256 (asymmetric: private key to sign, public to verify)

Common JWT Claims

Claim	Description
iss	Issuer
sub	Subject (usually user ID)
aud	Audience
exp	Expiration time (UNIX timestamp)
nbf	Not before (valid after this time)
iat	Issued at time

Claim	Description
jti	JWT ID (unique identifier)

How JWT Works in Authentication

1. Login: User sends credentials to server.

2. Issue Token: Server verifies credentials and issues a JWT.

3. Store Token: Client stores JWT (e.g., in localStorage or cookies).

4. Authenticated Requests: JWT sent in Authorization: Bearer header.

5. Validation: Server validates signature and expiration before granting access.

JWT Security Considerations

- Use HTTPS to avoid MITM attacks.
- Use HTTP-only, Secure cookies for refresh tokens.
- Don't store JWTs in localStorage (XSS risk).
- Keep tokens short-lived (exp).
- Never put sensitive data in payload.
- Use strong, rotating secrets or key pairs.

Node.js Example

```
require("dotenv").config();
const express = require("express");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");

const app = express();
app.use(express.json());



//    In-memory user store (for demo)
const USERS = [
  {
    id: 1,
    email: "user@example.com",
    password: bcrypt.hashSync("123456", 10) // hashed password
  }
];

//    JWT helper functions
const generateAccessToken = (user) => {
  return jwt.sign({ id: user.id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRY || "15m"
  });
};

const generateRefreshToken = (user) => {
  return jwt.sign({ id: user.id }, process.env.REFRESH_TOKEN_SECRET, {
    expiresIn: process.env.REFRESH_TOKEN_EXPIRY || "7d"
  });
};

//    Middleware: Verify access token
const authenticate = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1];
  if (!token) return res.sendStatus(401);

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.sendStatus(403); // Token expired or invalid
    req.user = user;
    next();
  });
};

//    Login Route
app.post("/api/login", (req, res) => {
  const { email, password } = req.body;
  const user = USERS.find(u => u.email === email);
  if (!user || !bcrypt.compareSync(password, user.password)) {
    return res.status(401).json({ message: "Invalid credentials" });
  }

  const accessToken = generateAccessToken(user);
  const refreshToken = generateRefreshToken(user);

  res.json({ accessToken, refreshToken });
});
```

```
//    Protected Route
app.get("/api/protected", authenticate, (req, res) => {
  res.json({ message: "Protected data", userId: req.user.id });
});

//    Refresh Token Route
app.post("/api/refresh", (req, res) => {
  const { refreshToken } = req.body;
  if (!refreshToken) return res.sendStatus(401);

  jwt.verify(refreshToken, process.env.REFRESH_TOKEN_SECRET, (err, user)
    if (err) return res.sendStatus(403);

    const newAccessToken = generateAccessToken(user);
    res.json({ accessToken: newAccessToken });
  });
});

//    Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`   Server running on http://localhost
```