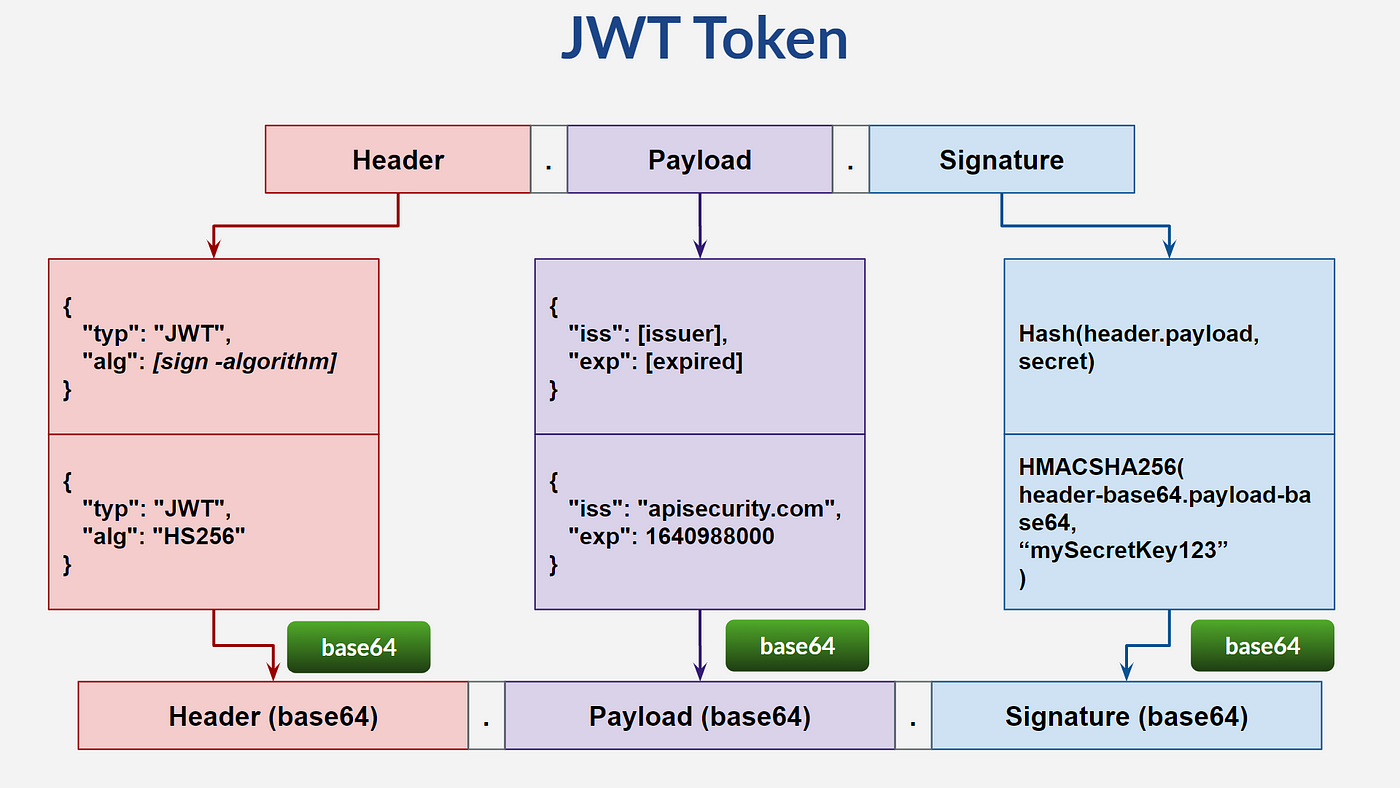
JWT UNDERSTADNING DOCUMENT

JSON Web Tokens (JWT) are a compact, URL-safe means of representing claims to be transferred between two parties.

typically found in authentication and authorization scenarios.



### **JWT Structure**

A JWT consists of three parts separated by dots (.):

1. **Header**
2. **Payload**
3. **Signature**

Example: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImpvaG5kb2UiLCJleHAiOjE1MTYyMzkwMjJ9.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

#### **1. Header**

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

Json

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

#### **2. Payload**

The payload contains the claims. Claims are statements about an entity (typically, the user) and additional metadata.

{ "username": "johndoe", "exp": 1516239022 }

#### **3. Signature**

To create the signature part, you have to take the encoded header, the encoded payload, a secret, and the algorithm specified in the header, and sign that.

HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

### **Real-time Example: User Authentication**

We'll use a Spring Boot application for this example.

#### **Step 1: User Logs In**

1. **User sends a POST request to /login with credentials:**

{ "username": "johndoe", "password": "password123" }

1. **Server verifies credentials:**
   * If valid, generate a JWT.
   * If invalid, respond with an error.

#### **Step 2: Generate JWT**

The server generates a JWT if the credentials are valid.

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;

public class JwtUtil {

private String secretKey = "your-256-bit-secret";

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10)) // 10 hours

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

}

#### **Step 3: Server Responds with JWT**

The server responds with the JWT:

{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImpvaG5kb2UiLCJleHAiOjE1MTYyMzkwMjJ9.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c" }

#### **Step 4: Client Stores JWT**

The client stores the JWT, usually in local storage or a cookie.

#### **Step 5: Client Makes Authenticated Requests**

The client includes the JWT in the Authorization header for subsequent requests.

Authorization: Bearer <JWT>

#### **Step 6: Server Validates JWT**

The server validates the JWT in each request.

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

public class JwtUtil {

private String secretKey = "your-256-bit-secret";

public Claims extractClaims(String token) {

return Jwts.parser()

.setSigningKey(secretKey)

.parseClaimsJws(token)

.getBody();

}

public boolean isTokenExpired(String token) {

return extractClaims(token).getExpiration().before(new Date());

}

public String extractUsername(String token) {

return extractClaims(token).getSubject();

}

}

Validate jwt

import io.jsonwebtoken.Claims;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

String authorizationHeader = request.getHeader("Authorization");

String token = null;

String username = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

token = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(token);

}

if (username != null && !jwtUtil.isTokenExpired(token)) {

// Set user authentication in security context

}

filterChain.doFilter(request, response);

}

}

### **Summary**

* **Login**: User sends credentials, server validates and returns a JWT.
* **Store JWT**: Client stores the JWT.
* **Authenticated Requests**: Client includes JWT in headers, server validates JWT for each request.

This flow ensures secure communication between client and server using JWT for authentication and authorization.

