JSON Web Token (JWT)

Last Updated : 11 Jul, 2025

A JSON Web Token (JWT) is a standard used to securely transmit information between a client (like a frontend application) and a server (the backend). It is commonly used to verify users' identities, authenticate them, and ensure safe communication between the two. JWTs are mainly used in web apps and APIs to protect against unauthorized access.

The data in a JWT, such as user details, is stored in a simple JSON format. To keep the data safe, the token is signed cryptographically, making sure that no one can alter it. The signing can be done using these cryptographic methods:

- HMAC (Hash-based Message Authentication Code)
- RSA or ECDSA (Asymmetric cryptographic algorithms)

JWTs are primarily used for authentication and secure data exchange in web applications and APIs.

How JWT token Works?

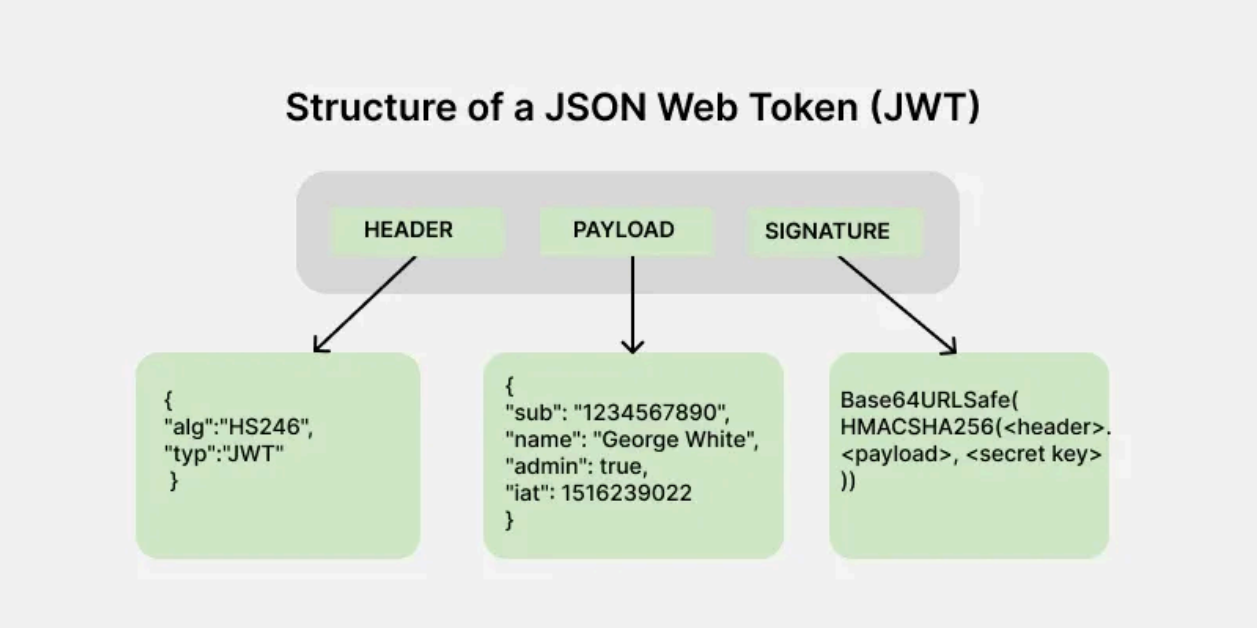
1. **User Logs In**: The client (browser) sends login credentials to the server.
2. **Server Generates JWT**: If credentials are valid, the server creates a JWT containing user data and signs it with a secret key.
3. **Token Sent to Client**: The JWT is sent back to the client and stored (usually in localStorage or a cookie).
4. **Client Sends Token in Requests**: For protected routes, the client includes the JWT in the Authorization header (Bearer Token).
5. **Server Verifies and Responds**: The server verifies the token, extracts user info, and processes the request if valid.

What are Tokens and Why Are They Needed?

Tokens are used to securely transmit sensitive information between the client and the server. Instead of sending plain data (e.g., user info) that could be

tampered with, tokens provide a secure method of validation. JWTs are widely adopted because they are tamper-proof, ensuring that data remains unaltered during transmission.

JWT Structure



Structure of a JWT

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

```
Header. Payload. Signature
```

1. **Header:** Contains metadata about the token, such as the algorithm used for signing.
2. **Payload:** Stores the claims, i.e., data being transmitted.
3. **Signature:** Ensures the token's integrity and authenticity.

1. Header

The header contains metadata about the token, including the signing algorithm and token type here metadata means data about data.

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

- **alg**: Algorithm used for signing (e.g., HS256, RS256).

- **typ**: Token type, always "JWT".

Base64Url Encoded Header

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

2. Payload

The payload contains the information about the user also called as a claim and some additional information including the timestamp at which it was issued and the expiry time of the token.

```
{
    "userId": 123,
    "role": "admin",
    "exp": 1672531199
}
```

Common claim types:

- **iss (Issuer):** Identifies who issued the token.
- **sub (Subject):** Represents the user or entity the token is about.
- **aud (Audience):** Specifies the intended recipient.
- **exp (Expiration):** Defines when the token expires.
- **iat (Issued At):** Timestamp when the token was created.
- **nbf (Not Before):** Specifies when the token becomes valid.

Base64Url Encoded Payload

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNzA4MzQ1MTIz
LCJleHAiOjE3MDgzNTUxMjN9
```

3. Signature

The signature ensures token integrity and is generated using the header, payload, and a secret key. In this example we will use HS256 algorithm to implement the Signature part

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

Example Signature:

```
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

4. Final JWT token

After all these steps the final JWT token is generated by joining the Header, Payload and Signature via a dot. It looks like as it is shown below.

> *eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNzA4MzQ1MTIzLCJleHAiOjE3MDgzNTUxMjN9.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c*

Security Considerations

- **Use HTTPS**: Prevent man-in-the-middle attacks by transmitting JWTs over HTTPS.
- **Set Expiration Time**: Prevent long-lived tokens that can be exploited.
- **Use Secure Storage**: Store JWTs securely (e.g., HttpOnly cookies instead of local storage).
- **Verify Signature**: Always validate the token's signature before trusting its content.

Implementing JWT in a web application

1. Code to create a JSON web token

This code generates a JWT (JSON Web Token) using the jsonwebtoken library in Node.js. The token contains user data and is signed with a secret key for security.

Command to install jsonwebtoken library in NodeJS

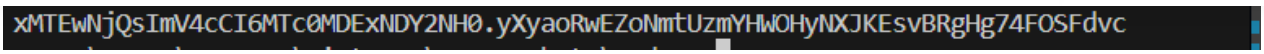
```
npm install jsonwebtoken
```

```
const jwt = require('jsonwebtoken');
const secretKey = 'abcde12345';

const token = jwt.sign({
  id: 1,
  username: 'GFG'
}, secretKey, { expiresIn: '1h' });

console.log(token);
```

Output



Code to create a JSON web token

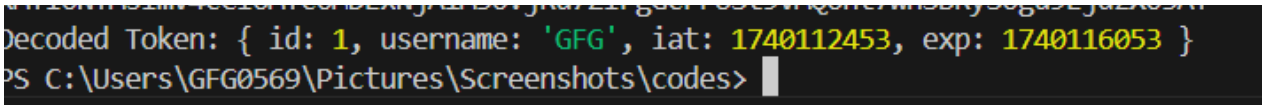
- **Importing JWT Library**: The jsonwebtoken module is required to create and verify tokens.
- **Defining Secret Key**: A secret key (abcde12345) is used to sign the token securely.
- **Creating JWT**: The jwt.sign() method generates a token with user details (id, username) and an expiration time of 1 hour.
- **Logging the Token**: The generated JWT is printed to the console for use in authentication.

2. Code to verify a JSON web token

This code verifies a JWT using the jsonwebtoken library in Node.js. It checks if the token is valid and extracts the payload if authentication succeeds.

```
jwt.verify(token, 'abcde12345', (err, decoded) => {
    if (err) {
      console.log('Token is invalid');
    } else {
      console.log('Decoded Token:', decoded);
    }
  });
```

Output



Code to verify a JSON web token

- **Verifying the Token**: The jwt.verify() method checks if the provided token is valid using the secret key.
- **Handling Errors**: If verification fails, an error (err) occurs, and "Token is invalid" is logged.
- **Decoding Token Data**: If valid, the decoded object contains the original user details.
- **Logging the Decoded Data**: The decoded payload is printed to the console, showing user details from the token.

Common Issues During Development with JWT

JWT errors often arise from mismatched details or token problems:

- **JWT Rejected :** This means the server couldn't verify the token. It might happen because:
    - **The JWT has expired**: The token is no longer valid because it passed its expiration time.
    - **The signature doesn't match**: The token might have been tampered with, or the signing keys have changed.
    - **Other claims don't match**: For example, if the token was created for one app but sent to another, the app will reject it because it doesn't match the expected details.
- **JWT Token Doesn't Support the Required Scope:** A JWT contains permissions (called "scopes") that define what actions the user has agreed to. If the app requires more permissions than the token provides, it will be rejected. For instance, if the app needs permission to modify data, but the token only allows reading data, it won't work.
- **JWT Decode Failed** : This happens when the token isn't in the expected format. For example, the client might expect the JWT to be base64 encoded, but if the server didn't encode it that way, the client won't be able to read it properly.

Advantages of using JSON Web Token

JWTs are widely used for authentication and authorization due to their numerous advantages:

- **Stateless Authentication**: No need to store user sessions on the server; JWT contains all necessary data.
- **Compact & Fast**: Being small in size, JWT is efficiently transmitted in HTTP headers, making it ideal for APIs.
- **Secure & Tamper-Proof**: JWTs are signed using a secret key or public/private key pair, ensuring integrity.
- **Cross-Platform Support**: Can be used with any technology (JavaScript, Python, Java, etc.) for authentication.
- **Built-in Expiry**: Tokens can have an expiration time (expiresIn), reducing the risk of long-term access misuse.

Conclusion

JSON Web Tokens (JWT) provide a secure, fast, and stateless way to handle authentication. They are widely used in APIs, web apps, and mobile apps due to their compact size, cross-platform support, and built-in security features. By leveraging JWT, developers can ensure safe and efficient user authentication without storing sessions on the server.

Similar Reads

How Long is a JWT Token Valid ?

JSON Web Tokens (JWTs) are widely used for authentication and authorization in modern web applications and APIs. One crucial aspect of JWTs is their validity period, which determines how long a token remains vali...

6 min read

Working with APIs in JavaScript

An API is simply a medium to fetch or send data between interfaces. Let's say you want to make an application that provides the user with some real-time data fetched from the server or maybe even allows yo...

5 min read

How to Create and Verify JWTs with Node?

In this article, we will see how to create JWT tokens in Node.js. We will implement secure authentication in Node.js by creating and verifying JSON Web Tokens (JWTs) using libraries like...

5 min read

How to securely handle sensitive data like passwords or tokens in Postman?

When dealing with sensitive data like passwords or tokens, it's crucial to follow best practices to ensure the security of your applications. In this guide, we will walk you through the step-by-step process of securely...

3 min read

Session-Based Authentication vs. JSON Web Tokens (JWTs) in System Design

Authentication is essential for websites and apps to verify users' identities. It's similar to showing your ID before entering a secured place or using a service online. There are two main ways to do this: Session-Based...

8 min read

JWT Authentication In Node.js

In modern web development, ensuring secure and efficient user authentication is paramount. JSON Web Tokens (JWT) offer a robust solution for token-based authentication, enabling secure transmission of user...

3 min read

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

OOAD

System Design Bootcamp

Interview Questions

Puzzles

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Commerce

World GK

GeeksforGeeks Videos

DSA

Python

Java

C++

Web Development

Data Science

CS Subjects