**Json Web Token Structure**

A JSON Web Token (JWT) is a JSON object that is used to securely transmit information between parties. The information can be verified and trusted, because it is digitally signed. JWT's are often used to authenticate users.

The structure of a JWT typically consists of three parts, separated by dots (.). These parts are:

1. Header: This contains information about how the JWT is encoded. It is a JSON object that typically consists of two fields: "alg" (algorithm) and "typ" (type).

2. Payload: This contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: registered, public, and private claims.

3. Signature: This is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way. The signature is created by taking the encoded header, the encoded payload, a secret, the algorithm specified in the header, and signing that.

Example:

Header: { "alg": "HS256", "typ": "JWT" } Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 } Signature: HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

JWT will be sent like : base64UrlEncode(header) + "." + base64UrlEncode(payload) + "." + signature

Please note that this is just a basic example and should not be used in production.

**Claims in Json Structure**

A JSON Web Token (JWT) is a JSON object that is used to securely transmit information between parties. The information can be verified and trusted, because it is digitally signed. JWT's are often used to authenticate users.

The structure of a JWT typically consists of three parts, separated by dots (.). These parts are:

Header: This contains information about how the JWT is encoded. It is a JSON object that typically consists of two fields: "alg" (algorithm) and "typ" (type).

Payload: This contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: registered, public, and private claims.

Signature: This is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way. The signature is created by taking the encoded header, the encoded payload, a secret, the algorithm specified in the header, and signing that.

Example:

Header:

{

"alg": "HS256",

"typ": "JWT"

}

Payload:

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022

}

Signature:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

JWT will be sent like :

base64UrlEncode(header) + "." + base64UrlEncode(payload) + "." + signature

Please note that this is just a basic example and should not be used in production.

**How are signatures verified in JWT**

In a JSON Web Token (JWT), the signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

The signature is created by taking the encoded header, the encoded payload, a secret, the algorithm specified in the header, and signing that. When the JWT is received, the signature is verified by performing the same process on the received JWT and comparing the resulting signature to the signature that was sent with the JWT.

The algorithm used to create the signature is specified in the "alg" field in the header. The most common algorithms used for creating JWT signatures are HMACSHA256 and RSA.

For example, if the algorithm specified in the header is "HS256", the signature is verified by:

1. Base64 decoding the header and payload.
2. Concatenating the decoded header and payload with a period (".") separator
3. Signing the resulting string using the HMACSHA256 algorithm with the secret specified in the server's configuration.
4. Comparing the resulting signature to the signature that was sent with the JWT. If they match, the JWT is considered to be valid.

It's important to note that the secret used to sign the JWT should be kept confidential on the server side, and never shared or exposed to the client.

It's also important to check the expiration time of the token(exp claim) and the not before time(nbf claim) to confirm that the token is not expired and can be used before verifying the signature.

Top of Form

**What form of Response do you get for JWT?**

When using JSON Web Tokens (JWT) for authentication, the server typically responds to a successful authentication request with an access token in the form of a JWT. This JWT is then sent by the client in the "Authorization" header of subsequent requests to the server, in order to authenticate the client.

The structure of the JWT returned by the server typically includes a header, payload, and signature, just like the JWT that was sent to the server for authentication. The header and payload contain information about the authenticated user, as well as other metadata such as the expiration time of the token, and the signature is used to ensure that the JWT has not been tampered with.

The JWT returned by the server can be in the form of a compact representation, which is encoded using base64Url encoding and concatenated using period separators.

Example of JWT response:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

The JWT can be passed in the HTTP headers, in the body of the request, or can be passed through a query string parameter, it depends on the implementation of the API or the client application.

It's important to note that the JWT should be stored securely on the client side, and should not be exposed to or shared with untrusted parties.

**Why are JWT headers necessary**

In a JSON Web Token (JWT), the header typically contains information about how the JWT is encoded. This information is used by the recipient of the JWT to determine how to process the JWT.

The two main fields in a JWT header are the "alg" (algorithm) field and the "typ" (type) field.

The "alg" field specifies the algorithm used to sign the JWT. This is important because it allows the recipient of the JWT to verify the signature and confirm that the JWT has not been tampered with. Common algorithms used for signing JWTs include HMACSHA256 and RSA.

The "typ" field is used to indicate the type of the token. It's optional but recommended to include. The default value is "JWT".

An example of a JWT header:

Copy code

{

"alg": "HS256",

"typ": "JWT"

}

The header can also include other fields, like "cty" (content type) or "kid" (key ID), which may be used for additional security or validation purposes.

In summary, the header in a JWT is necessary to specify the algorithm used to sign the JWT, and it allows the recipient to verify the signature and confirm that the JWT has not been tampered with. Additionally, it can also contain other fields that provide additional security or validation information.

**java 11 features**

Java 11, released in September 2018, introduced several new features and changes, including:

* Local-variable type inference, using the "var" keyword, which allows developers to declare a variable without specifying its type.
* The removal of the Java EE and CORBA modules, which were considered obsolete.
* The introduction of the "HttpClient" API, which provides a modern and efficient way to send HTTP requests.
* The ability to launch a single-file source-code programs without the need for a separate compilation step.
* The addition of the "var" keyword in lambda expressions, which allows for more concise and readable code.
* The ability to create a unmodifiable List, Set, and Map using the List.of, Set.of, and Map.of methods.
* The enhancement of GC(garbage collector) and JIT (Just-In-Time) compiler.
* The addition of the "jshell" tool, which allows developers to quickly test and run Java code snippets from the command line.

**java8 features**

* Lambda expressions, which allow for functional programming and increased use of parallelism.
* The Stream API, which allows for functional-style operations on collections of data.
* The Date and Time API (JSR 310), which provides improved date and time support.
* The Optional class, which helps to prevent NullPointerExceptions.
* The Nashorn JavaScript Engine, which allows Java applications to execute JavaScript code.
* The Base64 Encoding/Decoding feature, which provides an easy way to encode and decode binary data.
* The Concurrent Accumulators and StampedLock, which provides a better way to do atomic operations and lock-free operations.
* The parallel operations on collections, which allows for improved performance on multi-core systems.
* The method references and functional interface, which enhances readability and code style.
* The default methods in interfaces, which allows for easier evolution of APIs.