TokenHeader

August 17, 2026

TokenHeader

Kind: Interface

Source: `src/utils/jwt/jwt.ts`

Part of: Utils

`TokenHeader` defines the header data for a JWT token. It identifies the signing algorithm, fixes the token type as `'JWT'`, and includes a key identifier for key selection during verification.

Properties

Property	Type
`alg`	`SignatureAlgorithm`
`typ`	`'JWT'`
`kid`	`string`

Diagram

```
graph LR
  Header[TokenHeader] --> Algorithm[alg: SignatureAlgorithm]
  Header --> Type[typ: JWT]
  Header --> KeyId[kid: string]
  Header --> Token[JWT header]
```

Usage

```
import type { TokenHeader } from './utils/jwt/jwt';
import type { SignatureAlgorithm } from './utils/jwt/jwa';

declare function getSigningAlgorithm(): SignatureAlgorithm;
```

```
const header: TokenHeader = {
  alg: getSigningAlgorithm(),
  typ: 'JWT',
  kid: 'active-signing-key',
};
```

AI Coding Instructions

- Set `typ` to the exact literal `'JWT'`.
- Supply an `alg` value that matches the signing or verification configuration.
- Set `kid` to the identifier expected by the key lookup process.
- Keep header construction aligned with the JWT signing and verification code paths.

How it works

`TokenHeader` is an exported TypeScript interface for the decoded or constructed header portion of a JWT. It declares a required `alg` signature-algorithm field, plus optional `typ` and `kid` fields. `src/utils/jwt/jwt.ts:38-42`

- `alg` must have the `SignatureAlgorithm` type. That type accepts `HS256`, `HS384`, `HS512`, `RS256`, `RS384`, `RS512`, `PS256`, `PS384`, `PS512`, `ES256`, `ES384`, `ES512`, or `EdDSA`. `src/utils/jwt/jwa.ts:7-23`
- `typ`, when present, is the literal string `'JWT'`; `kid`, when present, is a string. `src/utils/jwt/jwt.ts:38-42`

JWT signing constructs a header with the selected algorithm and `typ: 'JWT'`. When the signing key is an object containing `alg`, signing replaces the function's algorithm argument with that key's `alg` value and also places the key's `kid` value in the header. The header is JSON-serialized, UTF-8 encoded, Base64URL encoded, and becomes the first dot-separated token part. `src/utils/jwt/jwt.ts:31-32` `src/utils/jwt/jwt.ts:56-75`

`isTokenHeader` is the runtime type guard associated with this interface. It returns `true` only for a non-null object that has an `alg` property whose value occurs in `AlgorithmTypes`, and whose `typ` property is either absent or exactly `'JWT'`. `src/utils/jwt/jwt.ts:44-54` It does not check that `kid` is a string, and it does not reject additional properties; its checks are limited to `alg` and `typ`. `src/utils/jwt/jwt.ts:45-51`

Decoded headers are cast to `TokenHeader` after Base64URL decoding and JSON parsing; `decode` and `decodeHeader` only reject tokens that do not have exactly three parts or whose decoding/parsing throws. Neither function calls `isTokenHeader`. `src/utils/jwt/jwt.ts:264-290`

Signature verification calls `isTokenHeader` and throws `JwtHeaderInvalid` when the decoded header fails that guard. It then requires `header.alg` to equal the caller-selected algorithm, throwing `JwtAlgorithmMismatch` otherwise. `src/utils/jwt/jwt.ts:123-129` JWK-set verification also validates the header, requires a truthy `kid` or throws `JwtHeaderRequiresKid`, rejects symmetric header algorithms, and requires the header algorithm to occur in `options.allowedAlgorithms`. `src/utils/jwt/jwt.ts:209-225`

Relationships

- IMPORTS → `decodeBase64Url`
- IMPORTS → `encodeBase64Url`
- IMPORTS → `AlgorithmTypes`
- IMPORTS → `signing`
- IMPORTS → `verifying`
- IMPORTS → `JwtAlgorithmMismatch`
- IMPORTS → `JwtAlgorithmNotAllowed`
- IMPORTS → `JwtAlgorithmRequired`
- IMPORTS → `JwtHeaderInvalid`
- IMPORTS → `JwtHeaderRequiresKid`
- IMPORTS → `JwtPayloadRequiresAud`
- IMPORTS → `JwtSymmetricAlgorithmNotAllowed`
- IMPORTS → `JwtTokenAudience`
- IMPORTS → `JwtTokenExpired`
- IMPORTS → `JwtTokenInvalid`
- IMPORTS → `JwtTokenIssuedAt`
- IMPORTS → `JwtTokenIssuer`
- IMPORTS → `JwtTokenNotBefore`
- IMPORTS → `JwtTokenSignatureMismatched`
- IMPORTS → `utf8Decoder`
- IMPORTS → `utf8Encoder`