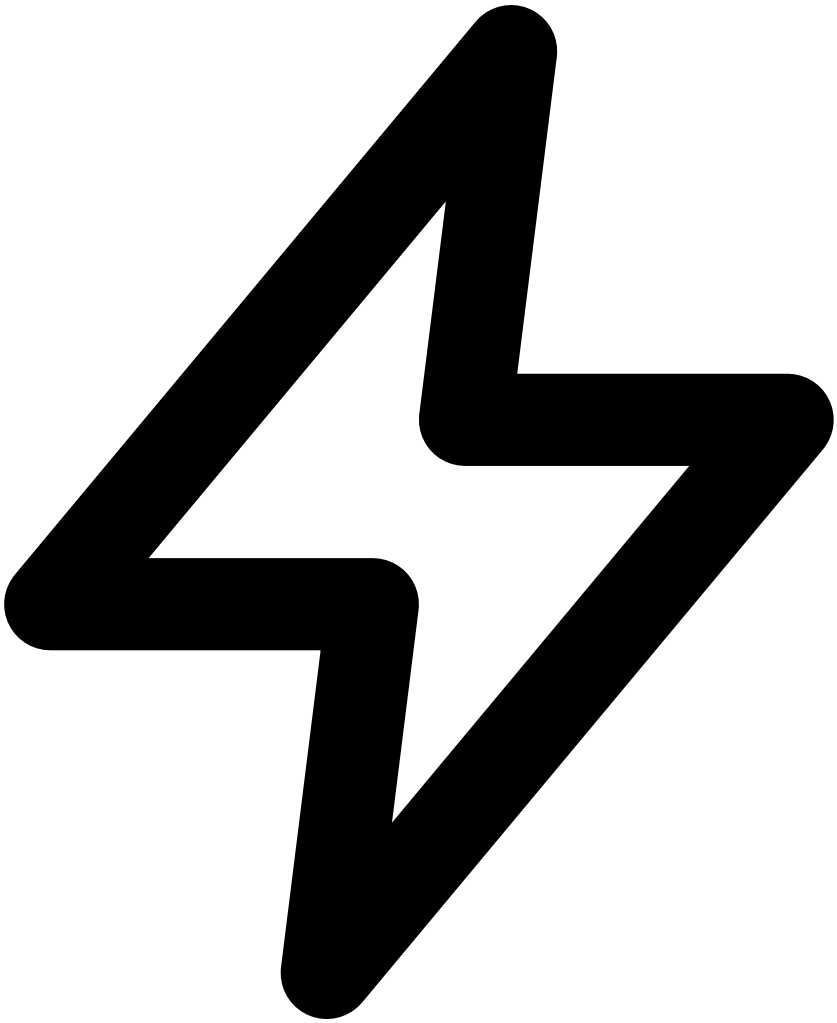

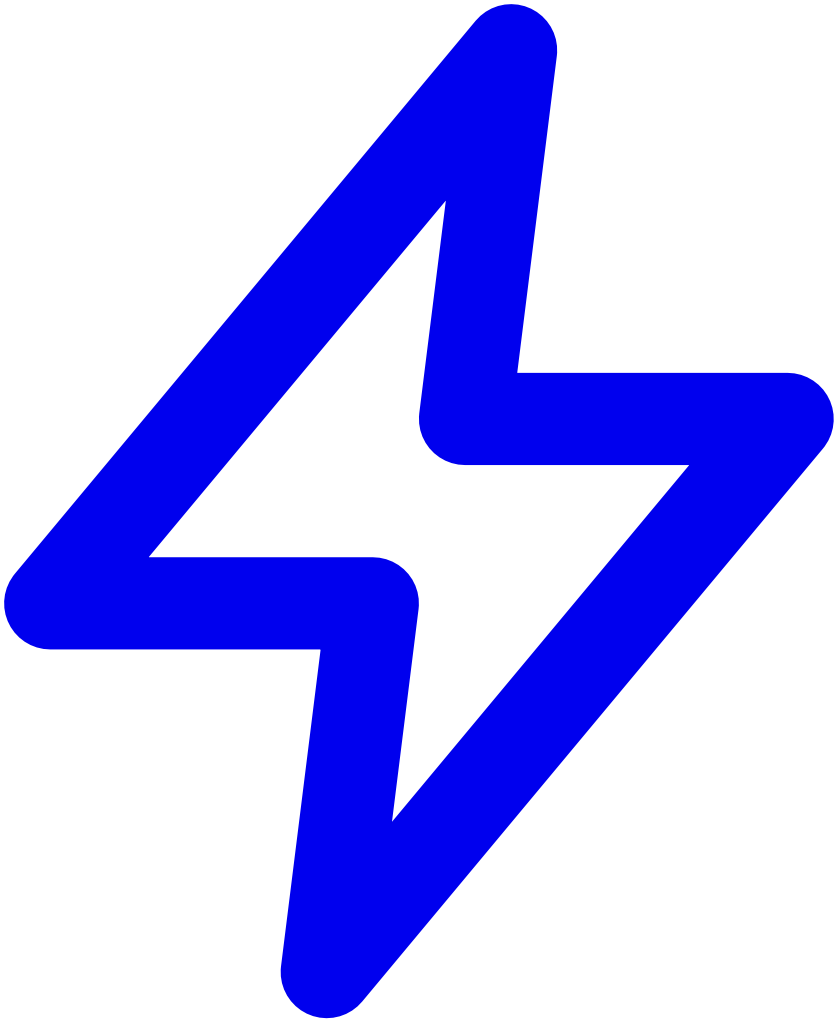

What Is a JSON Web Token (JWT)? A Plain-English Introduction

By the **AutonomiqTech Editorial Team** · Reviewed against our editorial standards · 8 min read · Last reviewed 2026

What Is a JSON Web Token (JWT)? A Plain-English Introduction

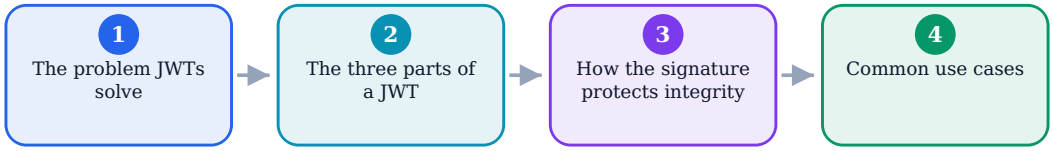

Figure: What Is a JSON Web Token (JWT)? A Plain-English Introduction

If you've worked with modern web APIs, you've probably encountered a long, cryptic-looking string called a **JSON Web Token**, or JWT. It's a compact way for systems to securely pass claims — often about who a user is — between each other.

This guide explains what a JWT is, how its three parts fit together, where it's used, and the security points that matter, all in plain English.

On this page

1. The problem JWTs solve
2. The three parts of a JWT
3. How the signature protects integrity
4. Common use cases
5. Security considerations that matter
6. Using JWTs responsibly
7. The anatomy of a JWT
8. What a JWT does and doesn't protect
9. Handling JWTs responsibly in practice

The problem JWTs solve

When a user logs in, the server needs a way to remember who they are on later requests without re-checking their password every time. One approach is a **token** the server issues after login, which the client sends back on each request. A JWT is a popular, standardised format for such tokens that's compact and self-describing.

The three parts of a JWT

A JWT looks like three chunks of text separated by dots: `header.payload.signature`. The **header** describes the token type and signing algorithm. The **payload** contains the claims — statements such as the user's ID and when the token expires. The **signature** is created from the header, payload and a secret, and lets the receiver verify the token is genuine.

The header and payload are just Base64-encoded JSON — readable by anyone — which

is a crucial point for security.

How the signature protects integrity

The signature is the clever part. The issuer combines the header and payload with a secret key and a signing algorithm to produce the signature. When the token comes back, the server recomputes the signature and compares it. If they match, the token is genuine and unaltered; if someone tampered with the payload, the signatures won't match.

This means a JWT can be trusted *even though anyone can read it*, because they can't forge a valid signature without the secret.

Common use cases

JWTs are widely used for **authentication** (proving who you are after login) and **authorization** (carrying permissions). They're popular in APIs and single-sign-on systems because they're self-contained: the server can validate the token and read the claims without a database lookup on every request.

Security considerations that matter

Two points catch people out. First, a standard JWT is **signed, not encrypted** — the payload is readable by anyone who has the token. Never put passwords or sensitive secrets in it. Second, because JWTs are self-contained and valid until they expire, **revoking** one early (say, on logout or a security event) isn't automatic and requires extra design, such as short lifetimes or a revocation list.

Using JWTs responsibly

Used well, JWTs are an elegant, scalable way to handle identity across services. Keep payloads free of secrets, use sensible expiry times, protect your signing key carefully, and transmit tokens over secure connections. Understanding both their strengths and their limits is the key to using them safely.

The anatomy of a JWT

A JWT's three parts each have a distinct job. Understanding them demystifies what looks like a random string:

Part	Contains	Purpose
Header	Token type and signing algorithm	Tells the receiver how it was signed
Payload	Claims (e.g. user id, expiry)	The actual information carried
Signature	A cryptographic signature	Verifies the token wasn't altered

The parts are joined by dots and encoded, which is why a JWT appears as three chunks separated by periods.

What a JWT does and doesn't protect

A common misunderstanding is what the signature actually guarantees:

- **It does** prove the token hasn't been tampered with and came from a trusted issuer.
- **It does not** hide the contents — a standard JWT payload is encoded, not encrypted, so anyone can read it.
- Therefore, never put secrets in a JWT payload.
- Always transmit tokens over secure connections and validate them on the server.

Handling JWTs responsibly in practice

Because JWTs are widely used for authentication and authorisation, handling them carelessly can create real security weaknesses, so a few responsible practices matter as much as understanding the format itself. Keep token lifetimes short so that a leaked token is only useful briefly, and pair short-lived access tokens with a sensible refresh strategy rather than issuing long-lived tokens that remain valid for weeks. Always verify the signature on the server before trusting any claim in the payload, and validate standard fields such as the expiry time so an old token is rejected automatically; skipping this check is a classic and dangerous mistake. Store tokens carefully on the client side, being mindful that different storage locations carry different risks, and always transmit them over encrypted connections so they can't be intercepted in transit. Because the payload is readable, treat it as public information and never place passwords, secrets or sensitive personal data inside it. Have a plan for revocation too, since a stateless token can't simply be ‘deleted’ server-side without additional mechanisms; short lifetimes, token blocklists or rotating signing keys are common approaches depending on the need. Finally, rely on well-maintained, established libraries to create and verify tokens rather than implementing the cryptography yourself, as subtle errors in signing or validation are easy to make and hard to spot. Followed together, these habits let you enjoy the convenience of JWTs — compact, self-contained, easy to pass between services — without inheriting the security problems that come from using them naively.

Printable checklist

Print this page or save the PDF to keep these steps handy.

- [ ] The problem JWTs solve
- [ ] The three parts of a JWT
- [ ] How the signature protects integrity
- [ ] Common use cases
- [ ] Security considerations that matter
- [ ] Using JWTs responsibly
- [ ] The anatomy of a JWT
- [ ] What a JWT does and doesn't protect

Summary

A JWT is a compact, self-contained token that carries claims (such as a user's identity) between parties. It has three parts — header, payload and signature — separated by dots. The signature lets the receiver verify the token wasn't tampered with. JWTs are popular for authentication and authorization, but they must be handled carefully: they're signed, not encrypted by default, and can't be easily revoked.

Key Takeaways

- A JWT is a compact token that carries claims, commonly used for authentication.
- It has three dot-separated parts: header, payload and signature.
- The signature verifies integrity — that the token wasn't altered.
- By default a JWT is signed, not encrypted, so don't put secrets in the payload.
- Because JWTs are self-contained, revoking them before expiry requires extra design.

Frequently Asked Questions

Is the data inside a JWT encrypted?

By default, no. A standard JWT is signed to prevent tampering, but the payload is only Base64-encoded and can be read by anyone with the token. Never store secrets in it unless you add encryption.

Can I invalidate a JWT before it expires?

Not automatically. Because JWTs are self-contained and valid until expiry, early revocation requires extra mechanisms such as short lifetimes, token blacklists, or a server-side session check.

What's the difference between the payload and the signature?

The payload holds the claims (like user ID and expiry). The signature is a cryptographic check, derived from the header, payload and a secret, that proves the token is genuine and unaltered.

Related Guides

- What Is an API?
- What Is Base64 Used For?
- HTTP Status Codes Explained
- UTF-8 vs ASCII

Related articles

- Regular Expressions (Regex) Basics: A Practical Starter Guide
- What Is an API? Explained Simply With Everyday Examples
- Hexadecimal Explained: Why Developers Count in Base 16