

what is a jwt , teach in tanglish for a beginner without any knowledge

Okay Vijis, let's break this down in **Tanglish style**, beginner level – no boring jargon.

### JWT na enna?

JWT = **JSON Web Token**.

Idhu oru *digital token* (small string of text) used in websites/apps to prove "nee yaara nu" (authentication) and sometimes "enna rights iruku" (authorization).

### Simple analogy:

Think like this –

- Nee oru **movie theatre** ku poita.
- Counter la ticket vangina. That ticket la un **seat number, show time, theatre name** etc irukkum.
- Inside theatre, ticket ah once show pannina, they won't keep checking your ID again and again. Ticket itself is the proof.

 JWT same madhiri work aagum. Once server gives JWT, adhu un ticket. Every request ku, nee andha JWT ticket ah server ku show panna pora. Server check pannitu, "seri, idhu authentic person da" nu allow pannum.

## Structure of JWT: (neraya interview la kekka chance irukkum 😊)

JWT ku 3 parts:

1. **Header** → info about token type & algorithm.
2. **Payload** → unga data (like user id, role, etc.).
3. **Signature** → secret code, to make sure no one altered it.

Example JWT:

xxxxxx.yyyyy.zzzzz

(idhu romba random text madhiri irukkum).

## Why JWT useful?

- Fast → server ku database check panna thevay illa each time.
- Portable → mobile app, web app anywhere use panna mudiyum.
- Secure → because of signature, JWT forge panna kashtam.

Sollu da, naan ipdi continue pannattuma deeper ah (JWT flow in login system, pros/cons), illa nee basic level ku stop panna solriya?

deep pa po , before that i have a doubt what is a token  
type in header and algorithm

payload , signature

Super question da, Vijis 🔥 — idhu thaan JWT oda core concepts. Naan ippo deep ah explain panren but still **Tanglish** la, so you won't get lost.

## ● 1. Token Type (in Header)

Header la first field usually:

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

👉 typ = Token type.

- Idhu oru **label** nu think panniko.
- Server ku sollradhuku, "hey, indha data JWT da, athoda rules follow pannu" nu.
- Athu mattum illa, vere type tokens (like JWE for encryption) irukkum, so JWT nu clear ah sollanum.

## ● 2. Algorithm (also in Header)

👉 alg = Algorithm.

- Idhu decide pannum, JWT oda **signature** epdi sign pannalaam nu.
- Example:
  - HS256 → HMAC with SHA-256
  - RS256 → RSA with SHA-256

So basically, alg = "math formula for locking the token."

### 3. Payload

👉 This is the core data inside the JWT.

Adhu la, app/server store panna info irukkum, example:

```
{ "sub": "1234567890", // subject = user id "name": "Vijis", "role":  
"student", "exp": 1694025600 // expiry time (in seconds) }
```

- **sub** = user ID
- **role** = unga role (student, admin, etc.)
- **exp** = expiry time (server ku sollanum, "idhu date ku appuram invalid").

👉 Payload la sensitive info (passwords) store panna koodadhu ✗, because JWT base64 encode pannum, encryption illa. Anyone can decode.

### 4. Signature

This is the tamper-proof seal.

- Server secret key use panni, Header + Payload ah join panni, algorithm apply pannum.
- Example:

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

👉 If someone edits payload (like role="user" → "admin"), signature mismatch aagum.

Server immediately detect pannidum.

## ✖ Final JWT Format

Header.Payload.Signature

Example:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlZpamlzIiwicm9sZSI6InN0dWR1bnQiLCJleHAiOjE20·  
SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

👉 In short:

- **Header** = rules (type + algorithm)
- **Payload** = info (user details, expiry, role)
- **Signature** = safety lock (proof no one changed it)

I can also show you **how login flow with JWT works** (step-by-step from login → token → request → verify). Want me to map that flow as a diagram explanation ah, illa text flow step-by-step ah?

ok from my understandning now i say , jwt have three parts token type where we tell the which type of token this is which which algorithm we use , have a doubt why we have to say jwt and need to enter the algo , payload jwt la eppadi use aago , signature la eppadi tamper proof aagithu , ennaku oru exmaple oda sollu from start to end the jwt process

Perfect da Vijis 🎉🔥 — you've got the structure nailed, now let's walk through JWT from start to end with a real example. I'll clear your doubts step by step.

#### ◆ 1. Why we say `JWT` and `alg` in Header?

- `typ: JWT` → Idhu ticket la "This is a movie ticket" nu print panni irukkaradhum madhiri. Server ku "Enna type token nu sollanum" illatti confuse aagidum. (E.g., JWT illa JWE nu?)
- `alg: HS256` → This is like telling "Ticket ink print pannirukkan, blue pen la sign pannirukkan." So server knows how to verify authenticity. Different algorithms = different math formulas.

#### ◆ 2. Payload eppadi use aagudhu?

Think: Movie ticket la **seat no, show time etc** irukkum. JWT payload la:

- `sub : user id`

- role : student/admin

- exp : expiry time

👉 Every time you show the token, server looks at the payload → "Okay, this is user 123, role student, valid till 8 PM."

### ◆ 3. Signature eppadi tamper proof aagudhu?

Oru simple trick:

Server does →

```
signature = HMACSHA256(base64(header) + "." + base64(payload), secretKey)
```

If hacker payload edit pannina (like role=student → admin ),

- The base64(payload) changes.
- So new signature generate aaganum.
- Hacker kitta secretKey illai.
- So server will reject it as "tampered."

👉 That's why signature is **tamper-proof lock**.

### ◆ 4. JWT Process Example (Start → End)

Let's say you log in to a college portal.

● Step 1: Login

- You enter: username = vijis , password = \*\*\*\*\* .
- Server checks database →  correct.

## ● Step 2: Server Creates JWT

Server builds:

Header

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

Payload

```
{ "sub": "101", "name": "Vijis", "role": "student", "exp": 1699455600 //  
expiry timestamp }
```

Signature (created using secret key: mysecret123 )

```
signature = HMACSHA256(base64(header) + "." + base64(payload), "mysecret123")
```

Final JWT =

aaaaaa.bbbbb.ccccc

## ● Step 3: Server Sends JWT

Server sends token back to you (browser/app). It's like giving you a **movie ticket**.

## ● Step 4: Client Stores JWT

Your app/browser saves it (usually in localStorage or sessionStorage).

## ● Step 5: Next Request

You ask: /get-my-marks .

Instead of sending username/password again, your app sends:

Authorization: Bearer aaaaaa.bbbbbbb.ccccccc

## ● Step 6: Server Verifies

1. Server splits JWT → header, payload, signature.
2. Recreates signature using header+payload+secret.
3. Compare with your signature → Match.
4. Checks exp (expiry). Still valid.
5. Reads role=student .
6. Returns your marks.

👉 If hacker changes payload role to admin , signature mismatch → Reject.

### ⚡ In short:

- Header = rules of token
- Payload = data of user
- Signature = guarantee "no one messed with it"
- Flow = Login → Token create → Send to client → Client sends it back → Server verifies  
→ Access granted

Do you want me to draw a diagram-like step flow (login → JWT creation → client → server) so it's super visual ah puriyum?

why we should use algo in this , ithu crt ta paru payload  
pathi athu namma details aah server ku soldrathu yaru  
login pandra enna time , ella update panna antha person  
na last ta verify panno,

and the secreat key na enna athu eppadi use aaguthu ?  
ipo reverify pannum both secret key poguthu apo kack  
panna mudiyatha

🔥 Nice da Vijis! You're not just learning, you're questioning like a true engineer — that's how you really understand security. Let's go point by point.

## ● 1. Why algorithm is needed?

Good doubt! Server ku payload verify panna **math formula** venum. That formula decide pannum:

- Epdi signature calculate pannuradhu?
- What cryptography use pannanum?

Example:

- HS256 → Uses **one secret key** for both signing & verifying.

- RS256 → Uses **private key** to sign, **public key** to verify.  
👉 Without telling "alg," server ku epdi verify pannuradhu nu theriyadhu. Like, nee paper la signature podra but adha check panna epdi? Pen ink ah? Digital pad ah? That's the alg.

## ● 2. Payload pathi correct ah sonna 😎

Yes, payload la namma info store pannuvom:

- sub : user id
- role : admin/student
- exp : expiry

But final authority = **signature + secret key**. Payload alone trustworthy illa (since anyone can decode and modify it).

## ● 3. Secret Key na enna?

Secret key = oru **server** ku mattum therinja password.

JWT signature epdi create aagudhu:

```
signature = HMACSHA256(base64(header) + "." + base64(payload), secretKey)
```

👉 **secretKey** ku onnu tha irukkum, adha hacker kitta illa.

## ● 4. Epdi verify pannudhu?

When client sends JWT back:

1. Server takes header + payload .
2. Re-runs same formula with `secretKey`.
3. Compares with JWT la irukkura signature.
  - If same → valid token.
  - If different → tampered! reject.

## 5. Apo hack panna mudiyatha?

Romba kashtam 🔒

- Hacker kitta payload visible irukkum (since base64).
- Hacker change pannalaam → but secretKey illa.
- So new valid signature generate panna mudiyadhu.
- Without secretKey, server accept pannadhu.

BUT if hacker somehow steals the `secretKey` (like bad server config, leaked code, weak key) → JWT totally broken. That's why secret key must be:

- Long, random, unguessable.
- Never shared outside server.

### ⚡ So final punchline:

- Algorithm = math method for locking/unlocking.
- Payload = info (but unsafe alone).

- Secret key = master password for signature.
- Reverify = server uses secret key → impossible to fake without it.