JWT Tokens Explained

JWT (JSON Web Token) is a standard for securely transmitting information between parties as a JSON object. Here's how they work:

Structure

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

- **Header**: Contains metadata about the token type and the signing algorithm used (like HS256 or RS256)
- **Payload**: Contains the actual data/claims - information about the user and additional metadata
- **Signature**: Ensures the token hasn't been tampered with

The format looks like: `header.payload.signature`

How It Works

1. **Authentication**: When a user logs in with valid credentials, the server creates a JWT containing user information and signs it with a secret key
2. **Token Storage**: The client receives the JWT and typically stores it (in memory, localStorage, or as an HTTP-only cookie)
3. **Subsequent Requests**: The client includes the JWT in the Authorization header of future requests: `Authorization: Bearer <token>`
4. **Verification**: The server verifies the signature using its secret key to ensure the token is valid and hasn't been modified
5. **Authorization**: If valid, the server extracts the user information from the payload and processes the request

Key Characteristics

Stateless: The server doesn't need to store session information - all necessary data is contained within the token itself.

Self-contained: The token carries all required information, reducing database lookups.

Expiration: JWTs typically include an expiration time (`exp` claim) after which they're no longer valid.

Claims: The payload contains claims - statements about the user (like user ID, email, roles) and additional metadata.

Security Considerations

The payload and header are Base64-encoded but not encrypted - anyone can decode and read them. The signature only ensures integrity, not confidentiality. Sensitive information shouldn't be stored in the payload unless the entire JWT is encrypted. The secret key used for signing must be kept secure, as anyone with the key can create valid tokens.

JWTs are particularly useful for distributed systems, microservices, and single sign-on scenarios where you need a standardized way to securely transmit user information between different services without maintaining server-side sessions.