# Token JWT

Prima di analizzare il meccanismo di autenticazione su JWT cerchiamo innanzitutto di capire cos’ è un JSON WEB TOKEN: abbreviato più comunemente con l acronimo di JWT. JWT è uno standard che viene usato per “regolare” le richieste tra due parti. JSON Web Token è un security token (una stringa)  che agisce come un container per le claims degli user. I claims sono informazioni di un utente: chi è l’utente loggato (e altre info di contorno: ad es mail, nome, cognome), scadenza di un token (quindi scadenza dell’autenticazione dell’utente), privilegi dell’utente (è un admin o un normale utente) ed altre info customizzabili a seconda del contesto applicativo . Tali informazioni sono firmate lato server secondo la specifica [JSON Web Signature (JWS)](http://self-issued.info/docs/draft-ietf-jose-json-web-signature.html), quindi il server è in grado di riconoscere se sono state generate da lui o meno. Riportiamo un esempio di token JWT:

1. eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImF1ZGllbmNlIjoid2ViIiwicm9sZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJdLCJpc0VuYWJsZWQiOnRydWUsImV4cCI6MTUxMjk3NTQ1MCwiaWF0IjoxNTEyOTY4MjUwMjMwfQ.J49qG\_yi0yMZP4K2PWddTNR7QyUloAR7qj\_3QtLL\_6A

Come si evince dall’esempio identifichiamo 3 parti in un token JWT

**Header** : Se decodifichiamo la prima parte del token scopriamo che si riferisce ad un json che contiene  le due voci “typ” e “alg”. Il primo ha come valore sempre “JWT”,  
mentre il nodo “alg” contiene il nome dell’algoritmo usato per il token.  
Nell’immagine sopra è stato usato l’algoritmo “HMAC-SHA256”.

**Payload** : La seconda parte del token è il corpo vero e proprio che contiene dei dati “variabili”  in base al contesto in formato json e codificati in base64. Nel nostro esempio il payload decodificato è

1. {
2. "sub": "admin",
3. "aud": "web",
4. "roles": [
5. "ROLE\_ADMIN",
6. "ROLE\_USER"
7. ],
8. "isEnabled": true,
9. "exp": 1512975450,
10. "iat": 1512968250230
11. }

JWT propone dei campi “standard” da utilizzare nel payload (es: sub, audience, exp e iat) ma il programmatore può inserire proprietà aggiuntive (vedi isEnabled e roles da me definiti e non contemplati nei clams standard riportato nel seguente [LINK](https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html#rfc.section.4.1.6)). Andando ad analizzare il json del body possiamo interpretare dal token le seguenti info:

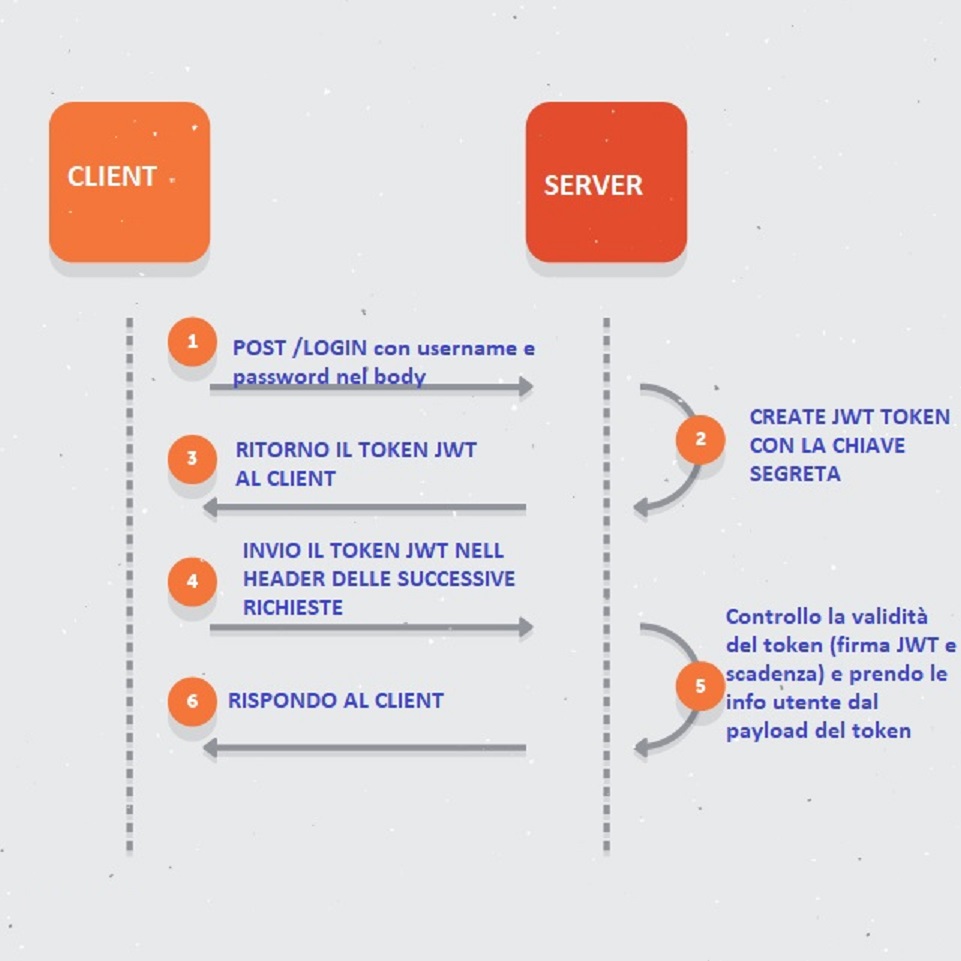
* utente admin (claim sub)
* ha come target il web  (claim aud)
* ruoli utente ROLE\_ADMIN, ROLE\_USER  (claim roles)
* utente abilitato  (claim isEnabled)
* info relative al momento in cui è stato creato il token (iat) e sulla data di scadenza (exp): momento in cui tale token non sarà piu’ ritenuto valido dal server.

**Signature**: La terza parte di un token è la firma che non è altro che il risultato di una funzione hash 256 che prende in input la codifica base64 dell’header concatenandola con un punto alla codifica base64 del payload, il tutto codificato con la nostra “chiave segreta” che solo il server conosce.

Con questa piccola introduzione sui JSON WEB TOKEN capiamo subito che questi token possono esser un ottimo modo per trasportare all’interno di ogni richiesta HTTP info legate all’autenticazione di un utente in modo totalmente sicuro: il server grazie al campo signature è in grado di capire che tali info sono state generate da lui grazie alla firma e che quindi garantiscono che tale richiesta  è da considerarsi relativa ad un utente autentificato.

### FLUSSO LOGICO DI AUTENTICAZIONE JWT

Ora che abbiamo introdotto i JSON WEB TOKEN descriverò come possono essere utilizzati per implementare il meccanismo di autenticazione in un backend RESTful. Il meccanismo di autenticazione JWT può essere riassunto in questo schema.



1. Il client, utilizzando l’endpoint di login, invia le proprie credenziali al server per autenticarsi.
2. Il server verifica se le credenziali postate sono corrette. In caso positivo costruisce un token jwt utilizzando la chiave segreta e inserendo nel payload del token le informazioni utente e le  data di creazione e scadenza del token; Tale token verrà firmato e autentificato con la sua chiave privata ( **importante solo il server deve conoscere la chiave privata** )
3. tale token viene inviato  nella risposta all’interno di un header http ( nella implementazione che proporremo sotto utilizzeremo l header http**X-auth**)
4. Il client salva il token jwt risultato dell’autenticazione, e nelle successive richieste richieste inoltra sempre nell’apposito header http  **X-auth**il token restituito nel punto 3. Tale token funge da “passaporto” per garantire che tali successive richieste sono autenticate.
5. **Il server riceve il token  ad ogni successiva richiesta. Ogni volta si  assicura che il token sia stato firmato e autentificato con la sua chiave privata ed infine che non sia scaduto.** Poiché il token contiene al suo interno il payload con le informazioni necessarie all’autenticazione (es. username e ruoli utente), il server eviterà di fare query ogni volta per verificare a quale utente corrisponde quel token e di recuperare le info utente necessarie per ricostruire lo stato della sessione (ottimo risultato per la scalabilità ed eliminando totalmente l’overhead della soluzione BASIC descritta ad inizio articolo).
6. Se passati i controlli descritti dal punto 5, il server tratterà l’utente come autenticato e gestirà la richiesta, altrimenti risponderà alla richiesta con un codice di errore HTTP 401 (utente non autorizzato).