

# Oauth 2

## Easy access delegation



# OAuth 2 rfc6749

## Si basa su un principio molto semplice:

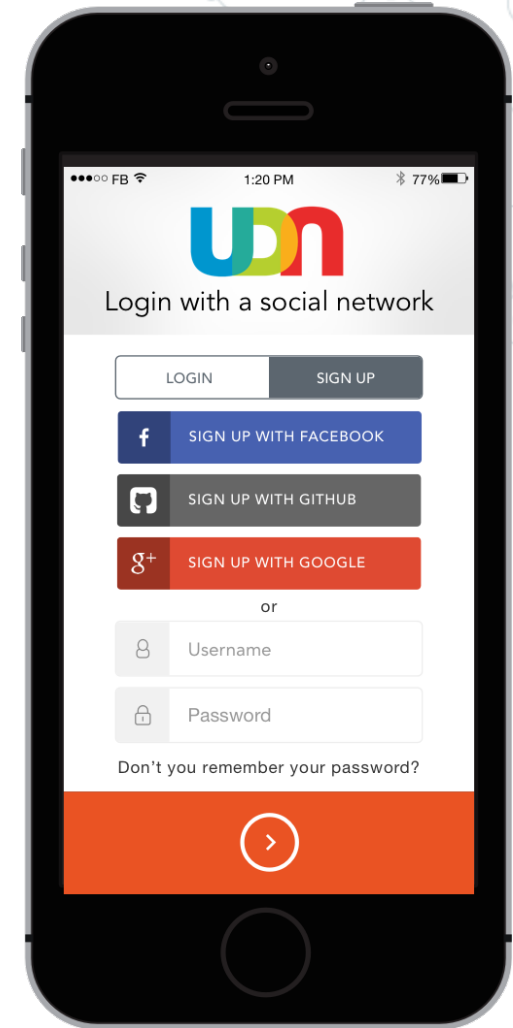
Garantire l'accesso ad applicazioni terze a delle risorse private senza condividere la propria password

## Perché non condividere la propria password?

- non si possono gestire livelli di autorizzazione differenti
- non si può garantire che l'autorizzazione venga utilizzata nel contesto scelto
- per revocare il permesso sono obbligato a cambiare password

OAuth è nato quindi con il presupposto di garantire l'accesso delegato ad un client specifico per determinate risorse sul server per un tempo limitato, con possibilità di revoca.

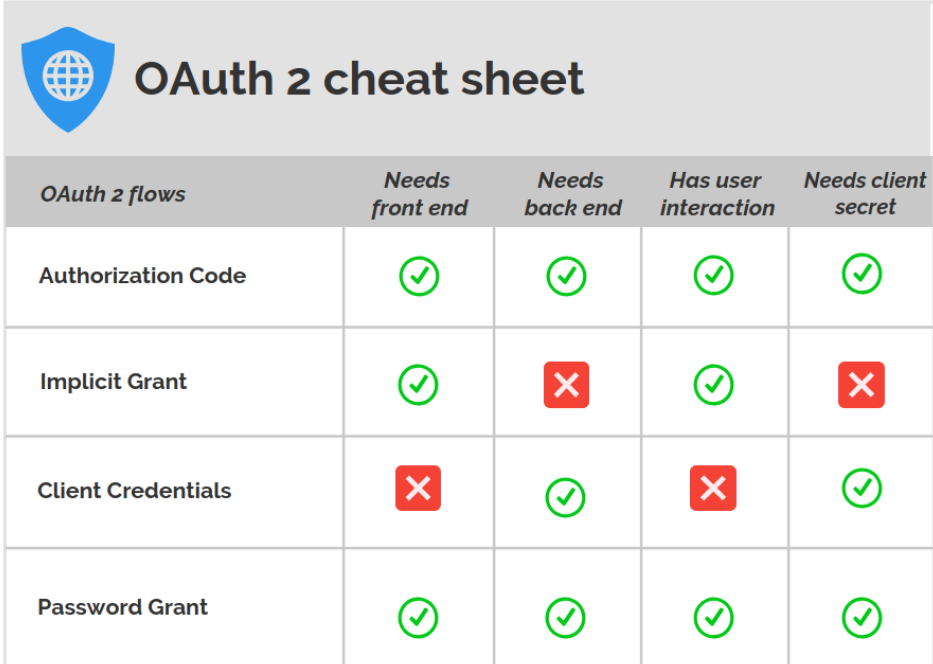
<https://it.wikipedia.org/wiki/OAuth>



# Oauth 2 rfc6749

Esistono differenti «flow» di autorizzazione descritti dal protocollo:

- Authorization Code Grant
- Implicit Grant
- Client Credential Grant
- Password Grant

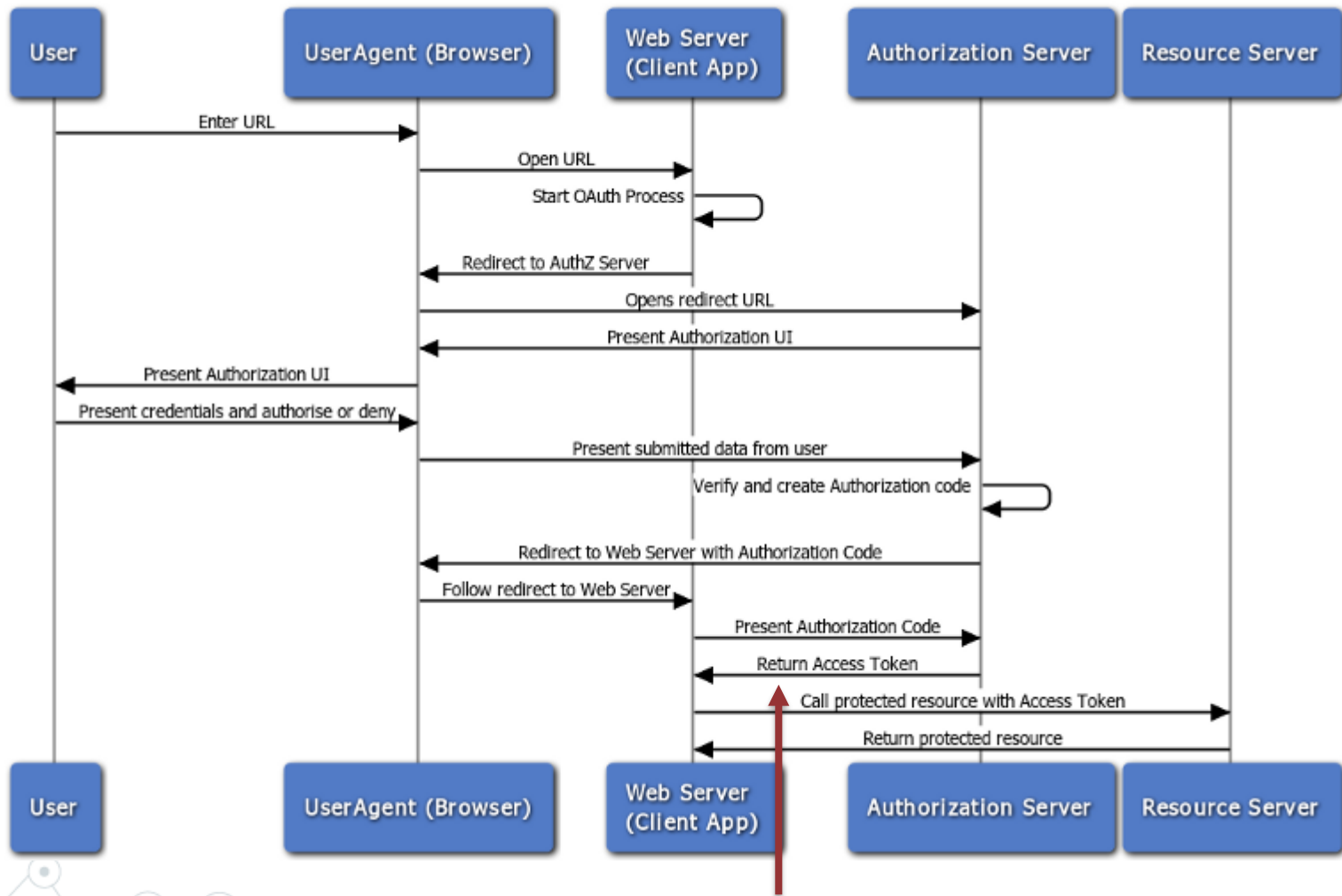


The image shows a cheat sheet for OAuth 2 flows. It features a blue shield icon with a globe inside, followed by the title "OAuth 2 cheat sheet". Below the title is a table with five columns: "OAuth 2 flows", "Needs front end", "Needs back end", "Has user interaction", and "Needs client secret". The rows represent the four grant types: Authorization Code, Implicit Grant, Client Credentials, and Password Grant. Each cell in the table contains a green checkmark (✓) or a red X (✗) indicating the requirements for that flow.

<i>OAuth 2 flows</i>	<i>Needs front end</i>	<i>Needs back end</i>	<i>Has user interaction</i>	<i>Needs client secret</i>
Authorization Code	✓	✓	✓	✓
Implicit Grant	✓	✗	✓	✗
Client Credentials	✗	✓	✗	✓
Password Grant	✓	✓	✓	✓

<https://itnext.io/an-oauth-2-0-introduction-for-beginners-6e386b19f7a9>

# Oauth 2: Authorization Code Flow



Bearer Tokens are the predominant type of access token used with OAuth 2.0. A Bearer Token is an opaque string, not intended to have any meaning to clients using it.

## Oauth 2: Complicato?



<https://auth0.com/pricing/>



Firebase Authentication

<https://firebase.google.com/pricing>

<https://auth0.com/blog/ionic-framework-how-to-get-started/>

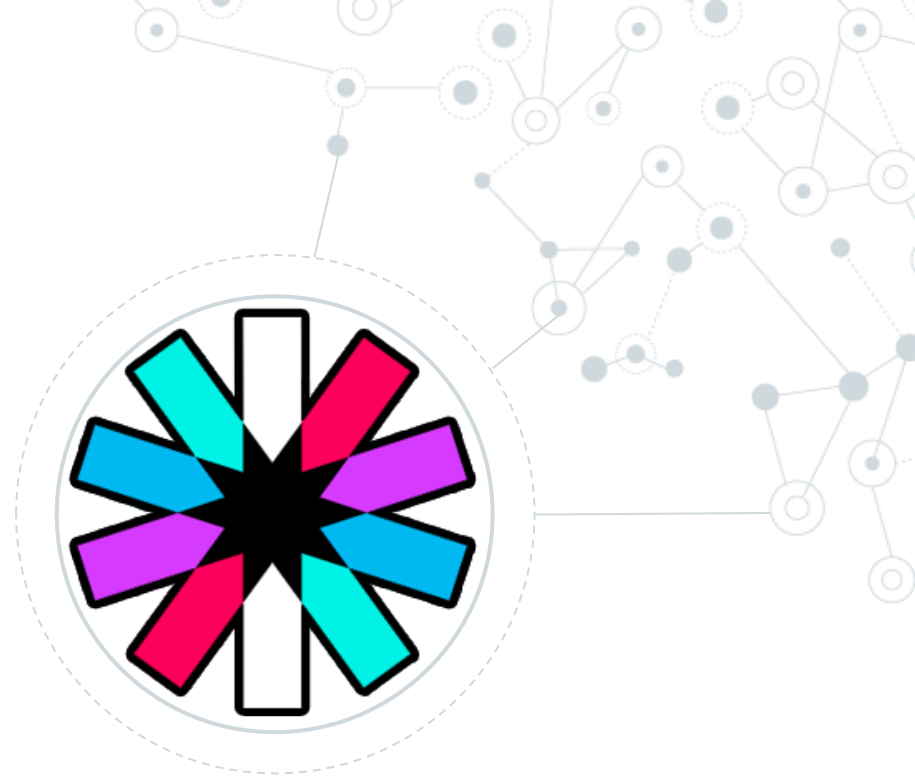
<https://auth0.com/docs/quickstart/spa/angular2/01-login>

<https://github.com/angular/angularfire>

<https://github.com/angular/angularfire/blob/master/docs/auth/getting-started.md>

# JWT

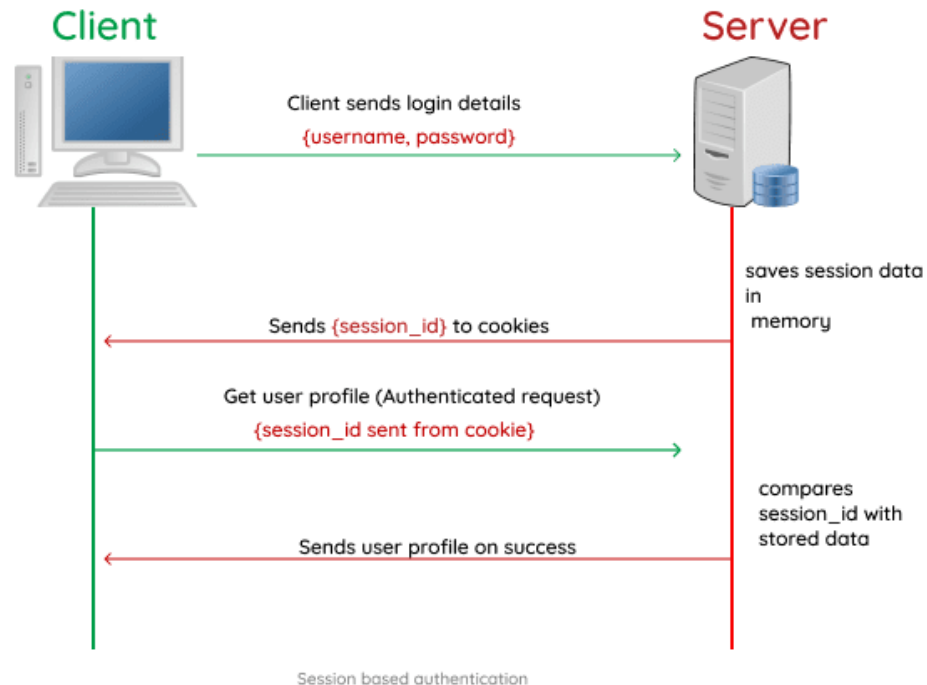
JWTs can be used as OAuth 2.0 [Bearer Tokens](#)



# JSON Web Token (JWT) rfc7523

## Perché è stato introdotto?

### Utilizzo classico delle sessioni



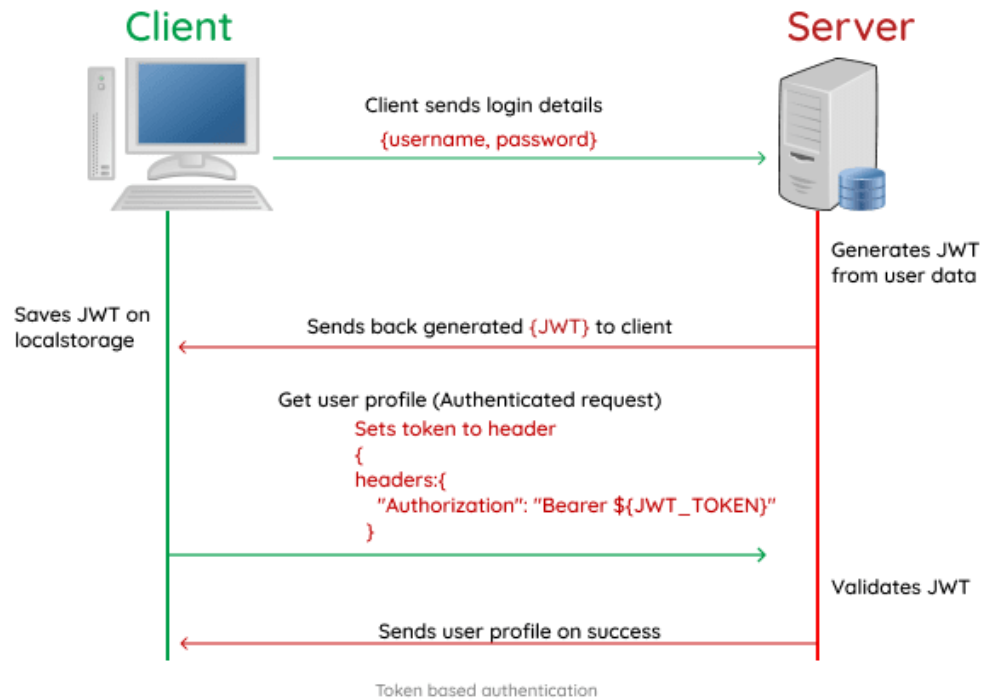
### Svantaggio delle sessioni:

- Devo andare sul database a verificare la validità della sessione utente
- Nelle architetture a microservizi l'uso della sessione non è agevole
- Non scalabile con architetture a cluster
- Su una SPA ho un maggiore tempo di sviluppo rispetto ad architetture server MVC

# JSON Web Token (JWT) rfc7523

## Perché è stato introdotto?

### Utilizzo del token



### Differenze?

- Non ho storage di sessioni
- Non devo gestire sessioni «morte» lato server
- I servizi possono essere realmente statefull

Ok ma dove è la vera differenza?



# JSON Web Token (JWT) rfc7523

<https://jwt.io/>

Cosa posso mettere nel JWT?

- Informazioni riguardante l'utente
- Informazioni sui permessi dell'utente
- Una data di scadenza
- Una signature per validare il JWT
- Qualsiasi altra informazione

**NIENTE PASSWORD NEL JWT!**

JWT  
JSON WEB TOKEN



HEADER  
ALGORITHM  
& TOKEN TYPE

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

+

PAYLOAD  
DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

+

SIGNATURE  
VERIFICATION

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),secretKey)
```

NORDICAPIS.COM

Lo scopo di un JWT non è di crittografare i dati, quindi evitare la lettura di dati sensibili durante il trasporto (esiste SSL), ma consente alla parte ricevente di fidarsi che i dati ricevuti sono rimasti inalterati durante il trasporto.