

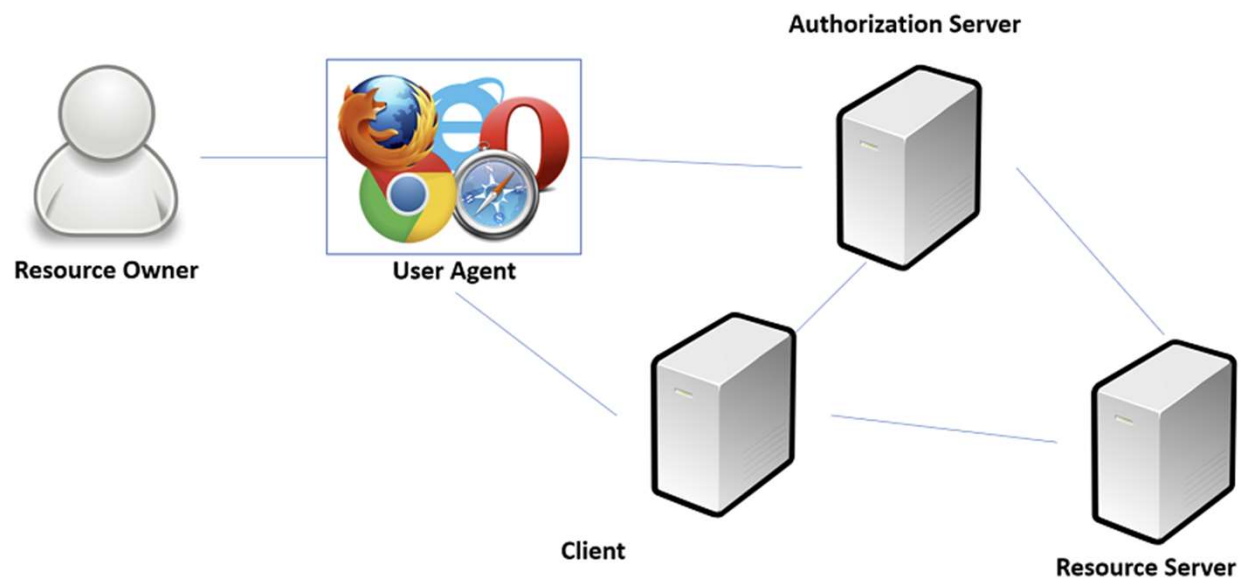
Autenticazione e autorizzazione con Oauth2 e il server Keycloak

Uso di JSON Web Token per gestire l'accesso alle risorse

Sommario

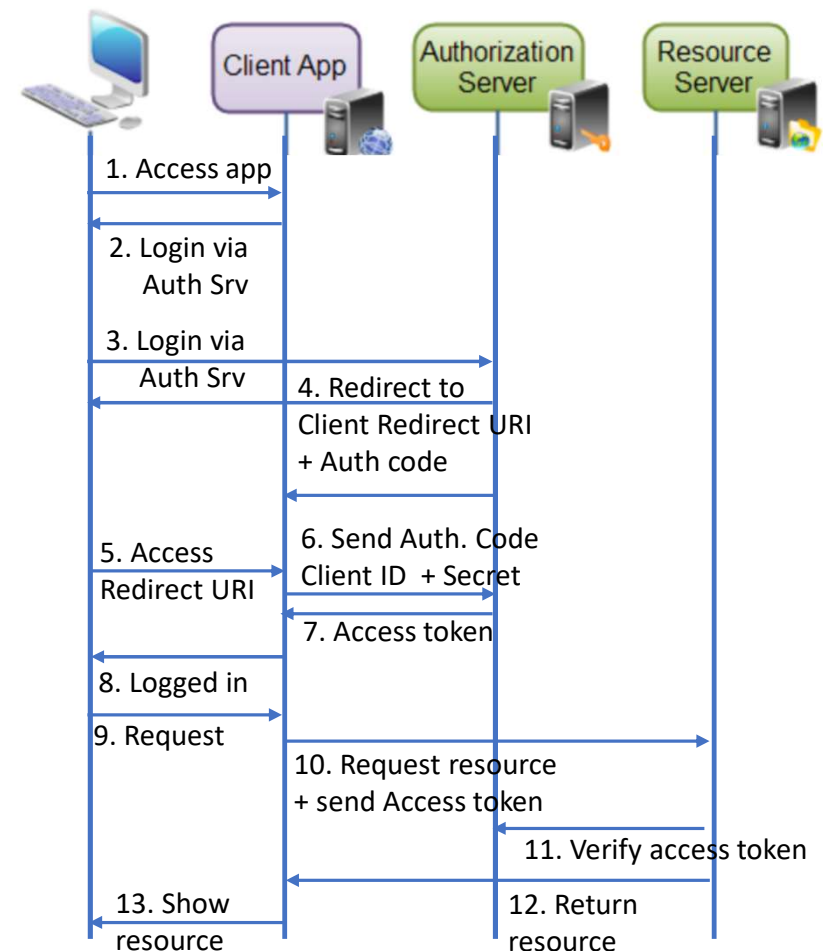
- Il protocollo OAUTH2
- JSON Web Token
- Il server Keycloak
- Applicazione al progetto

Richiamo sulla terminologia



Il protocollo

1. Il proprietario delle risorse (user) accede all'applicazione client.
2. L'applicazione client rinvia l'utente al login presso l'auth. server.
3. Per il login l'utente è ridiretto all'authorization server.
4. Quindi l'utente si autentica con l'authorization server. Se autenticato gli viene chiesto se intende concedere le risorse richieste all'applicazione. Se l'utente conferma è rinvio alla applicazione client (via Client Redirect URI) e con un Auth.Code
5. Il rinvio avviene mandando l'utente alla "redirect URI", specificato all'atto della registrazione. Insieme al rinvio, l'authorization server manda un codice che rappresenta l'autorizzazione (Auth. Code)
6. Quando si accede all'URI nell'applicazione client questa si connette direttamente all'authorization server. E invia il codice di autorizzazione, il client ID (ed eventualmente il segreto)
7. Se l'authorization server li ritiene validi ritorna un *token d'accesso*.
8. Lo user riceve conferma e può iniziare a fare le sue richieste alla client application (9)
10. L'applicazione client può ora usare il token per chiedere accesso alle risorse al resource server. Il token vale sia come autenticazione del client e del proprietario delle risorse (user), che come diritto di accesso alle risorse.
11. Il resource server può verificare la validità del token, in caso positivo risponde restituendo la risorsa (12, 13)



JSON Web Token (1/2)

- Il token usato per l'accesso può assumere diversi formati, uno dei più diffusi è il JWT – JSON Web Token (si pronuncia «jot»)
(per approfondimenti vedere jwt.io dove sono anche indicate varie librerie per la decodifica dei token, l'estrazione delle informazioni contenute nonché la verifica dell'autenticità del token)
- Il JWT comprende tre parti (codificate in BASE64 – base64url3):
 - Header
 - Payload (che potrà contenere varie informazioni, tra cui il tipo di permessi di accesso concessi con questo token)
 - Signature/Encryption data
- Il Resource Server può controllare l'autenticità del token verificando che il token sia firmato dall'Authorization Server (tramite la sua chiave pubblica).

JSON Web Token (2/2)

ESEMPIO:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

DECODIFICATI:

| | |
|---|---|
| Header: { "alg": "HS256", "typ": "JWT" } | Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 } |
|---|---|

Collegandosi a jwt.io e inserendo il JWT sopra, potete verificarne l'autenticità inserendo la parola *secret* nel campo Verify Signature.

Un esempio di Authorization Server: Keycloak

- Keycloak è un authorization server che si può scaricare e sperimentare localmente
(per approfondire <https://www.keycloak.org/>)
- Potete attivare una istanza locale sulla vostra macchina ed utilizzare quella per testare il sistema

NOTE: Le ultimissime versioni hanno subito alcune modifiche, in particolare per quanto riguarda gli endpoint (le URL da utilizzare per accedere alle diverse funzioni del server) – quanto indicato di seguito è stato testato con keycloak versione 12 che è ancora disponibile per il download. I cambiamenti dovrebbero tuttavia riguardare solo le versioni più recenti, e gli esempi valere per versioni fino alla 17.

Un testo esplicativo con esempi



<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Keycloak – Identity and Access Management
for modern Applications

Stian Thorgersen, Pedro Igor Silva

2021 Packt Publishing

Alcune funzionalità di Keycloak

- Permette di definire dei «realms» corrispondenti a contesti all'interno dei quali si possono registrare delle web-applications (la Client-App del diagramma di sequenza) che si appoggiano a keycloak (Authorization server nel diagramma di sequenza) per l'autenticazione (degli utenti) e per autorizzare l'accesso alle risorse (fornisce i token).
- Alle Client-App registrate si possono associare degli «scopes» che definiscono il tipo di permessi di accesso che saranno concessi dal Resource server (Es: potrebbe esserci uno scope di sola lettura, e uno che invece ha anche il permesso di modificare).
- All'interno di ogni *realm* si possono anche creare alcuni utenti ai quali possono essere associati ruoli per un controllo più fine degli accessi.

Installare keycloak

- Scaricare keycloak (per esempio la versione messa a disposizione sul dir) keycloak-12.0.4.zip
- Estrarre i file dallo zip
- Creare un utente amministratore (KC_HOME è il pathname della directory dove è stato estratto il file .zip)

To create an admin account on Linux or macOS, execute the following command in a terminal:

```
$ cd $KC_HOME
```

```
$ bin/add-user-keycloak.sh -u admin -p admin
```

On Windows, execute the following command:

```
> cd %KC_HOME%
```

```
> bin\add-user-keycloak.bat -u admin -p admin
```

Avviare keycloak su localhost

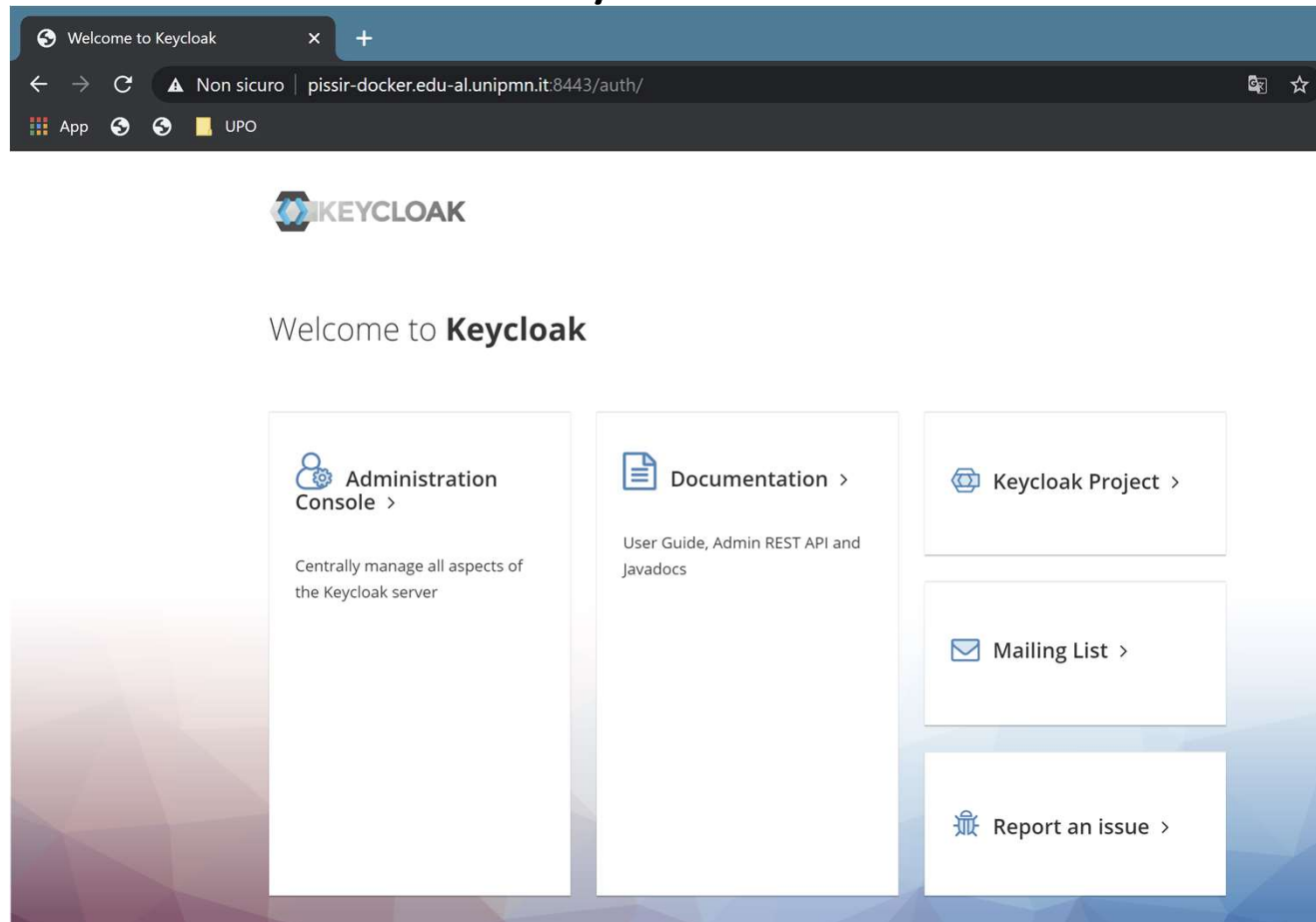
`.\bin\standalone.bat` oppure `./standalone.sh`

In questo modo il server viene avviato sulla porta 8080; se si vuole cambiare la porta su cui risponde si può aggiungere un parametro; per esempio per lanciare il server sulla porta 8443 ($8080 + 363 = 8443$)

`./standalone.sh -Djboss.socket.binding.port-offset=363`

Aprendo sul browser `localhost:8080/auth` si accede alla interfaccia di amministrazione

Interazione con Keycloak: la console di amministrazione



Quando installate keycloak per prima cosa occorre creare un utente admin che può **configurare i realms e registrare client applications** ed **utenti**.

I *realms*: contesti all'interno dei quali si attivano delle applicazioni (le client-app) e si registrano degli utenti

esempio di keycloak lanciato su localhost

The screenshot shows the Keycloak Admin Console interface in a web browser. The address bar indicates the URL is `localhost:8443/auth/admin/master/console/#/realms/myrealm`. The left sidebar contains a navigation menu with the following items: **Myrealm** (selected), **Master**, **MyDemo**, **Add realm**, **Client Scopes**, **Roles**, **Identity**, **Providers**, **User Federation**, **Authentication**, and **Manage**. The main content area is titled **Myrealm** and features a tabbed interface with the following tabs: **General** (active), **Login**, **Keys**, **Email**, **Themes**, **Cache**, **Tokens**, **Client Registration**, and **Security Defenses**. The **General** tab displays the following configuration fields:

- Name**: `myrealm`
- Display name**: `MyRealm`
- HTML Display name**: `MyRealm`
- Frontend URL**: (empty field)
- Enabled**: **ON** (toggle switch)
- User-Managed Access**: **OFF** (toggle switch)
- Endpoints**: `OpenID Endpoint Configuration`

The screenshot shows the Keycloak administration interface. On the left is a dark sidebar with navigation links: Myrealm, Configure (Realm Settings, Clients, Client Scopes, Roles, Identity, Providers, User Federation, Authentication), and Manage (Groups, Users, Sessions, Events, Import). The main content area is titled 'Clients > clientid-00' and shows the configuration for 'Clientid-00'. The 'Settings' tab is active, displaying fields for Client ID (clientid-00), Name (ClientTrial), Description (Just to try it), Enabled (ON), Consent Required (OFF), Login Theme, Client Protocol (openid-connect), Access Type (confidential), and Standard Flow Enabled (public bearer-only). The 'Access Type' dropdown is open, showing 'confidential' as the selected option.

Clientid-00

Settings Credentials Roles Client Scopes Mappers Scope Revocation Sessions

Offline Access Clustering Installation

Client ID clientid-00

Name ClientTrial

Description Just to try it

Enabled ON

Consent Required OFF

Login Theme

Client Protocol openid-connect

Access Type confidential

Standard Flow Enabled public bearer-only

esempio di keycloak lanciato su localhost

Se il client è *confidential* c'è un «secret» (campo *Credentials*)

Le client-app (continua esempio)

just to try it

☒ ON ☐ OFF

openid-connect

confidential

Standard Flow Enabled ?

☒ ON ☐ OFF

Implicit Flow Enabled ?

☐ OFF

Direct Access Grants Enabled ?

☒ ON ☐ OFF

Service Accounts Enabled ?

☐ OFF

Authorization Enabled ?

☐ OFF

http://localhost:8080/

Description ?

Enabled ?

Consent Required ?

Login Theme ?

Client Protocol ?

Access Type ?

Root URL ?

* Valid Redirect URIs ?

Base URL ?

Qui si vede la URI dove viene
rinviato lo user una volta
completata la fase di
autenticazione

KEYCLOAK

Roles

Admin

Myrealm

Configure

Realm Settings

Clients

Client Scopes

Roles

Identity

Providers

User Federation

Authentication

Manage

Groups

Users

Roles

Realm Roles

Default Roles

Search...

Q

View all roles

Add Role

| Role Name | Composite | Description | Actions | |
|-------------------|-----------|----------------------------|---------|--------|
| new_role | False | Ruolo di prova | Edit | Delete |
| offline_access | False | \${role_offline-access} | Edit | Delete |
| uma_authorization | False | \${role_uma_authorization} | Edit | Delete |

KEYCLOAK

Users

Myrealm

Configure

Realm Settings

Clients

Client Scopes

Roles

Identity

Lookup

Search...

Q

View all users

| ID | Username | Email | Last Name | First Name | Actions |
|---------------------|---------------|-----------------------|-----------|------------|---------|
| 14f4df9e-3da8-47... | myrealm_user2 | giuliana.francesch... | Cognome | Nome | Edit |
| 8924b047-a536-4... | myuser | giuliana.francesch... | User | My | Edit |

Interrogare keycloak per ottenere JWT (confidential)

Vediamo alcune interazioni con keycloak via *curl* (caso di client-app *confidential*)

```
curl -X POST http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect/token -H 'content-type: application/x-www-form-urlencoded' -d 'username=clarabella&password=claraPWD2000&grant_type=password&client_id=ClientID-00&client_secret= 4db97bce-6aec-4998-ba6a-f454b1c51560'
```

Si ottiene un JWT:

```
{"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXZWQ6IiwiaWF0IjoxNjU2aOdZVh6gCrd60iuTA","expires_in":3600,"refresh_expires_in":1800,"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWQ6IiwiaWF0IjoxNjU2aOdZVh6gCrd60iuTA","token_type":"bearer","not-before-policy":0,"session_state":"7e647a4c-346a-4ebe-88da-dddfa1bb29c5","scope":"email profile"}
```

(se anziché *confidential* la client app fosse *public* non si inserirebbe il *secret* nella richiesta)



```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "ZrgXZkehxCjleg770Es8xxGcYJ5R1qD1jICARH0dxv8"
}
```

PAYLOAD: DATA

```
{
  "exp": 1683878953,
  "iat": 1683878653,
  "jti": "691ef2b9-3ab5-4a2b-b356-f5cb32d9c73c",
  "iss": "http://localhost:8080/auth/realms/H2Orealm",
  "aud": "account",
  "sub": "d47ff083-b79d-4611-a483-1765e4936279",
  "typ": "Bearer",
  "azp": "ClientID-00",
  "session_state": "813b84cb-a16c-4f70-a5f0-2ffdf195ebeac",
  "acr": "1",
  "allowed_origins": [
    "http://localhost:8080"
  ],
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "email profile",
  "email_verified": false,
  "name": "ClaraBella Giglioli",
  "preferred_username": "ClaraBella",
  "given_name": "ClaraBella",
  "family_name": "Giglioli",
  "email": "giuliana.franceschinis@uniupo.it"
}
```

VERIFY SIGNATURE

RSASHA256(

Qui si possono vedere i contenuti decifrati dell'header e del payload.

https://it.wikipedia.org/wiki/Bas_e64

Verificare il JWT: ottenere la chiave pubblica

```
curl http://localhost:8080/auth/realms/H2Orealm
```

```
{"realm":"H2Orealm","public_key":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAg  
hTb8oN6ee5r/n0uccVVHfrfEaAAr34rGgMleBC3Y/nVezZv1VdQj6AWVuuGThGQF+YKit3A5+MT  
2qyhXQNNQ+Xv20ijeika6X2za7OFWMz7AlltNOBZamiUQjg4mN6eOxgcw/vrtIQJTKauFsiun47Q2  
Dv4oxyEmd79N+BdRTm9693om0Ub7WZT2RryWPQO5drPEvwcMtqOs9CCy8bdaSpPTruxJRSq  
1UJs68f1XZVpKWGjkHSzc3bpmGwVxJklt58PZx9D20P64t4SnUFWi1mxBjxu5uqEKPS2wRB2OO  
QTxf4NSFZIMrYyGF43M64qnehJTHPOveE4bl6WxNjm8AwIDAQAB","token-  
service":"http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect","account-  
service":"http://localhost:8080/auth/realms/H2Orealm/account","tokens-not-before":0}
```

```
----- BEGIN PUBLIC KEY -----
```

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgThb8oN6ee5r/n0uccVVHfrfEaAAr34  
rGgMleBC3Y/nVezZv1VdQj6AWVuuGThGQF+YKit3A5+MT2qyhXQNNQ+Xv20ijeika6X2za7OFW  
Mz7AlltNOBZamiUQjg4mN6eOxgcw/vrtIQJTKauFsiun47Q2Dv4oxyEmd79N+BdRTm9693om0  
Ub7WZT2RryWPQO5drPEvwcMtqOs9CCy8bdaSpPTruxJRSq1UJs68f1XZVpKWGjkHSzc3bpmGw  
VxJklt58PZx9D20P64t4SnUFWi1mxBjxu5uqEKPS2wRB2OOQTxf4NSFZIMrYyGF43M64qnehJTH  
POveE4bl6WxNjm8AwIDAQAB
```

```
----- END PUBLIC KEY -----
```

Verifica della signature

lJpmYwXzZSwibmFtZSI6K5vDWUgQ29nbm9tZSIs
InByZWZlcnJlZF91c2VybmFtZSI6Im15cmVhbG1f
dXNlcjIiLCJnaXZlbnl9uYW1lIjoIIm9tM9tZSIsImZh
bWlseV9uYW1lIjoIIm9tM9tZSIsImVtYWlsIjoI
Z211bG1hbmEuZnJhbmNlc2NoaW5pc0BkaS51bmlw
bW4uaXQ1fQ. ZokRPlj1NuoyQZyU9ZOUfXj9A9C6m
IZx8-W7s6fMqJ5sB-u6Dg4fXbncjoIVvpp-
CQR1E8DM9HbFMXppGaszczH8uMQkFGSwfmFsM-
4jaRyr_iaSR2Yq-
rPrKsN1Uu3QM9aLHBM46Q5YugvPQThYq1R_ipQMZ
mC2Ln_JEZ541FcDquDX1sgfcHPuApG-Us14-
KSDT5SA3cd8SsTNW1SiKxeJkXUnmpvAhKsjGkG0P
EDODDQXMd8LEHWFKecPtndhb19LIFgBS--
u_BPnqDms5sBKnyMbT-
N5ToivdbDLYmwUw7oJuyEcONBcLZQ0f4N7MMoV56
a0dZVh6qCrd60iuTA

```

"resource_access": {
  "account": {
    "roles": [
      "manage-account",
      "manage-account-links",
      "view-profile"
    ]
  }
},
"scope": "email profile",
"email_verified": false,
"name": "Nome Cognome",
"preferred_username": "myrealm_user2",
"given_name": "Nome",
"family_name": "Cognome",
"email": "giuliana.franceschinis@di.unipmn.it"
}

```

VERIFY SIGNATURE

```
RSASHA256(  
  base64UrlEncode(header) + " " +  
  base64UrlEncode(payload),  
  DJE0FRd9DiedilodseJ9oRQL02w  
  NfxXz/dsw5e4zSJ7TE0VQ9C5o8J  
  AInJvEtpwIDAQAB  
  -----END PUBLIC KEY-----  
  
Private Key. Enter it in plain  
text only if you want to gener  
ate a new token. The key never  
leaves your browser.
```

SHARE JWT

Signature Verified

Inserita la chiave pubblica nel campo «public key» della sezione «VERIFY SIGNATURE» vediamo che è stato validato il JWT

Nota: la chiave dev'essere preceduta da

-----BEGIN PUBLIC KEY-----

e seguita da

-----END PUBLIC KEY-----

Osservazioni sul JSON contenente la chiave pubblica:

Sono indicati:

l'indirizzo dove richiedere il token (usato in precedenza in richiesta token):

"token-service":

"http://localhost:8080/auth/realms/H2Orealm/protocol/openid-connect",

l'indirizzo per l'autenticazione degli utenti registrati:

"account-service":

"http://localhost:8080/auth/realms/H2Orealm/account"