

## OPEN ID CONNECT

Grazie a questo standard (estensione di OAuth2.0 che mira a rendere disponibile servizi di autenticazione) i servizi su web possono delegare la gestione dell'autenticazione degli utenti ad un servizio esterno, con una semplificazione dei servizi che ne fanno uso, e riducendo il numero di credenziali che un utente deve gestire e il numero di volte che deve effettuare l'autenticazione.

Sono oramai molti i siti (e servizi) web che permettono di autenticarsi con l'account di google o dei social network. Vale la pena osservare che in ambito aziendale ciò consente il single-sign-on per tutti i propri sistemi.

UN SECONDO PROGRAMMA DI ESEMPIO PER ANALIZZARE IN DETTAGLIO TUTTI I PASSAGGI DEL PROTOCOLLO DI AUTENTICAZIONE VIA OpenID Connect e acquisizione del token per l'autorizzazione come previsto dallo standard OAuth2.0

Il programma è nella cartella ch4 del pacchetto scaricabile da github (associato al libro [1]):

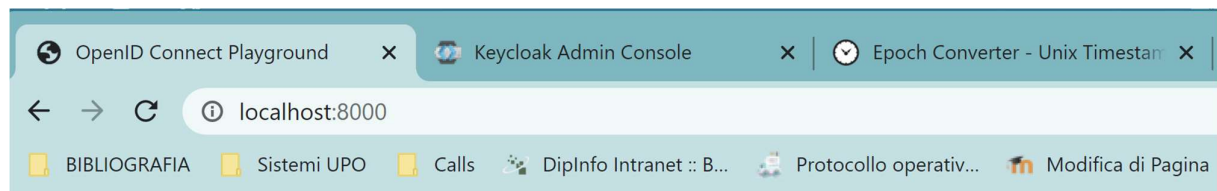
<https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications>

Il programma si lancia in Node JS spostandosi nella cartella ch4 e poi richiamando da linea di comando

```
npm install
```

```
npm start
```

Il servizio così attivato si raggiunge su <http://localhost:8000> e si presenta la seguente UI:



## OpenID Connect Playground

1 - Discovery 2 - Authentication 3 - Token 4 - Refresh 5 - UserInfo Reset

### Discovery

Issuer

### OpenID Provider Configuration

Il primo passo "Discovery" permette di ricavare una serie di metadati che forniscono informazioni utili sugli endpoint di keycloak, sui tipi di protocolli autorizzativi (grant type) supportati e sugli algoritmi utilizzabili per le firme digitali che vengono apposte sui token.

L'informazione che si ottiene cliccando sul pulsante Load OpenID Provider Configuration è la stessa che si otterrebbe accedendo dal browser direttamente all'endpoint "Issuer" mostrato nella casella di testo: vediamo alcuni estratti:

```
"authorization_endpoint": "http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth",  
  
"token_endpoint": "http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token",  
  
"introspection_endpoint": "http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token/introspect",  
  
"userinfo_endpoint": "http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/userinfo",  
  
"end_session_endpoint": "http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/logout",
```

Questi endpoint sono quelli da utilizzare per l'autenticazione e l'autorizzazione, per verificare il token, per ottenere informazioni sull'utente che si è autenticato, per chiudere una sessione (logout).

Per quanto riguarda i tipi di autorizzazione previsti, alcuni sono:

```
"grant_types_supported": [  
  "authorization_code",    <<<<<<<< flusso di tipo authorization code  
  ...  
  "client_credentials" ],  
"response_types_supported": [  
  "code",    <<<<<<<< authorization code restituito  
  ...  
  "id_token", <<<<<<<< token contenente le informazioni sull'utente autenticato  
  "token",    <<<<<<<< access token – per poter accedere a qualche risorsa dell'utente  
  ... ],
```

Per quanto riguarda gli algoritmi per la firma digitale dei token, sono elencati i seguenti:

```
"token_endpoint_auth_signing_alg_values_supported": [  
  "PS384", "ES384", "RS384", "HS256", "HS512", "ES256", "RS256", "HS384", "ES512", "PS256", "PS512", "RS512"  
],
```

In aggiunta vengono elencati gli *scopes* supportati:

```
"scopes_supported": [  
  "openid",  
  "address",  
  "email",  
  "microprofile-jwt",  
  "offline_access",  
  "phone",  
  "profile",  
  "roles",  
  "web-origins" ],
```

Ora esperimenterebbero i vari passaggi della sequenza di tipo “authorization code”:

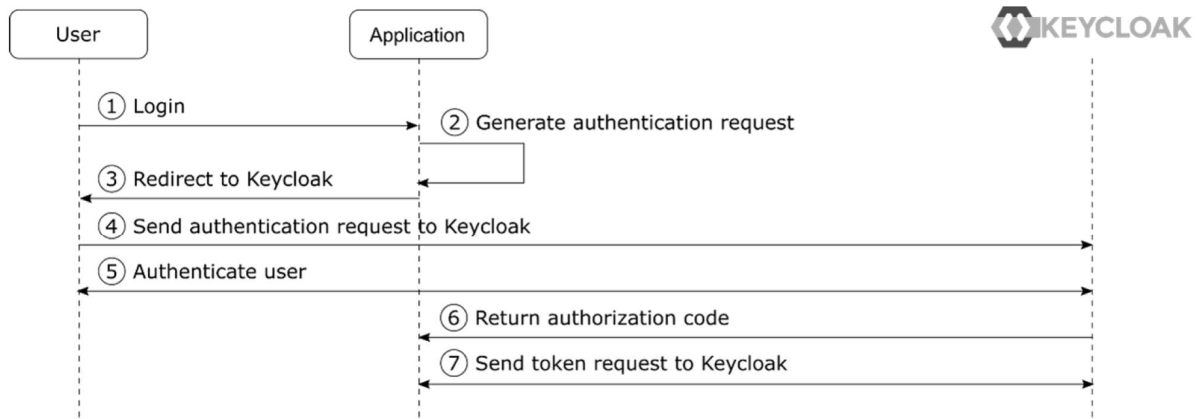


Figure 4.3 – The authorization code flow

Per generare la richiesta di autenticazione cliccare sul bottone 2-Authentication:

## OpenID Connect Playground

1 - Discovery 2 - Authentication 3 - Token 4 - Refresh 5 - UserInfo Reset

### Authentication

client\_id oidc-playground

scope openid

prompt

max\_age

login\_hint

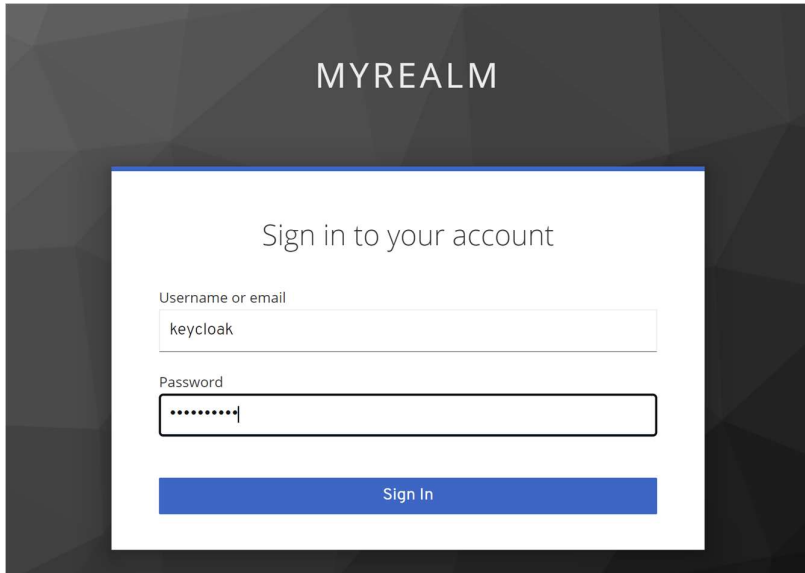
Generate Authentication Request

### Authentication Request

Send Authentication Request

### Authentication Response

Inserendo il nome di una client-application (qui se ne sta usando una ad-hoc per questo esperimento, chiamata oidc-playground ) e l'indicazione "scope id" = openid si può poi cliccare su "generate authentication request" per vedere cosa verrà inviato a kekcloak (passo 2 del diagramma di sequenza nella figura 4.3) e poi a seguire "send authentication request" (passo 4) che porta su Keycloak per l'inserimento delle credenziali:



Si possono osservare due cose: nell'authentication request si usa l'endpoint "authorization\_endpoint" ricavato dai metadata, e nella richiesta si specificano il nome della client-application che effettua la richiesta, il tipo di risposta che ci si attende (code) e la URI a cui ridirigere il controllo una volta completato l'inserimento delle credenziali.

## Authentication Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/auth  
  
client_id=oidc-playground  
response_type=code  
redirect_uri=http://localhost:8000/  
scope=openid
```

In risposta alla richiesta, dopo l'avvenuto inserimento delle credenziali si ottiene l'authorization code che ora la client application potrà reinviare a Keycloak per ottenere in cambio ID token ed eventualmente un access token (e refresh token associato)

## Authentication Response

```
code=fc89ccc9-bcad-4bda-a691-4c67c7d4100e.6cd8c160-65c6-41c5-82b6-7a53b0c7a241.2088eac8-450a-4071-a031-76bddf99acb6
```

Passando ora alla sezione 3-Token request si può ottenere un token – attenzione perché l'authorization code ha una scadenza molto breve e se si lascia passare più di un minuto dalla authentication request si ottiene una risposta del seguente tipo:

## Token Response

```
{  
  "error": "invalid_grant",  
  "error_description": "Code not valid"  
}
```

Se si invia l'authorization code rapidamente dopo averlo ricevuto, si ottiene il token: notiamo che si è usato il "token\_endpoint", indicando come grant\_type "authorization\_code" più informazioni sull'id della client app che sta effettuando la richiesta.

## Token Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token

grant_type=authorization_code
code=563310e2-71f6-4934-beb1-538badc1256a.6cd8c160-65c6-41c5-82b6-7a53b0c7a241.2088eac8-450a-4071
client_id=oidc-playground
redirect_uri=http://localhost:8000/
```

## Token Response

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHhRUnpIMTZuTDdwd1hLWldGbz...",
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYzI4OTI4OS01OGQzLTRmZDEt...",
  "token_type": "Bearer",
  "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIxLUQyMHhRUnpIMTZuTDdwd1hLWldGbz...",
  "not-before-policy": 0,
  "session_state": "6cd8c160-65c6-41c5-82b6-7a53b0c7a241",
  "scope": "openid profile email"
}
```

La versione decodificata (base64-url) dell'ID token (che è un JWT) è mostrata sotto nella finestra: riconosciamo le tre sezioni del token: Header.Payload.Signature.

Nel Payload possiamo vedere varie informazioni tra le quali la scadenza ("exp") del token (che possiamo convertire in data e ora inserendolo per esempio in (<https://www.epochconverter.com/> )

Si vede a quale "realm" si fa riferimento (campo "iss") e il nome della client application che ha fatto la richiesta ("azp"), i ruoli che potranno essere usati per limitare eventualmente l'accesso da parte della client application, e poi un certo numero di dati anagrafici dell'utente che si è appena autenticato.

Nota: queste sono le stesse informazioni che si potrebbero ottenere inserendo la stringa dell'access\_token nella finestra "Debug" del sito [jwt.io](http://jwt.io)

Per rinnovare un token scaduto, andando nella sezione 4-Refresh token, si può inviare una richiesta per ottenere un nuovo token (senza far ripetere l'inserimento delle credenziali all'utente). La richiesta differisce dalla precedente nell'authorization\_grant che in questo caso è refresh\_token, e nell'invio del refresh\_token al posto del code.

Infine si può sperimentare l'uso dell'endpoint "userinfo" che restituisce le stesse informazioni sull'utente che sono anche contenute nell'ID token:

---

## UserInfo Request

```
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/userinfo
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUI
```

## UserInfo Response

```
{
  "sub": "92777220-d719-431d-a161-faf72862c2ba",
  "email_verified": false,
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "myrole",
      "newrole"
    ]
  },
  "name": "Giuliana Annamaria Franceschinis",
  "preferred_username": "keycloak",
  "given_name": "Giuliana Annamaria",
  "family_name": "Franceschinis",
  "email": "giuliana.franceschinis@gmail.com",
  "picture": "https://upobook.uniupo.it/Files/People/284/e5"
}
```

Keycloak permette di aggiungere nuovi attributi e ruoli agli utenti, e nuovi scopes alla client application.

## BIBLIOGRAFIA:

[1] *Keycloak – Identity and Access Management for modern Applications*, Stian Thorgersen, Pedro Igor Silva, 2021 Packt Publishing

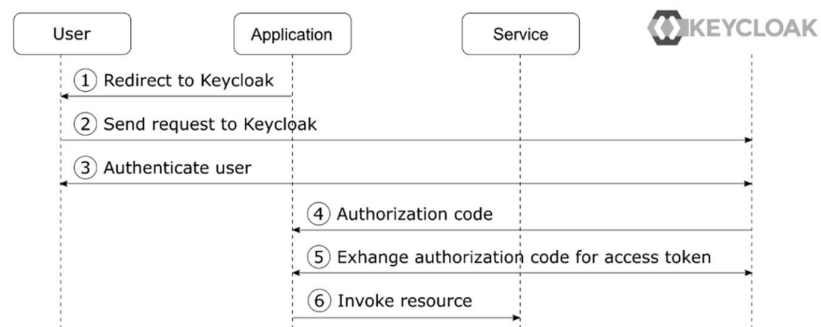


Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified