**Guida Progetto Keycloak OAuth 2.0 – Autenticazione e Autorizzazione API**

**1. Obiettivo del Progetto**

Lo scopo del progetto è dimostrare un flusso di autenticazione e autorizzazione informatica basato su OAuth 2.0, utilizzando Keycloak come Identity Provider e proteggendo API REST sviluppate con Flask.

La dimostrazione consiste in:

* Ottenere un Token JWT da Keycloak.
* Accedere a una API Libera (senza autenticazione).
* Accedere a una API Protetta (che richiede il token).

**2. Tecnologie Utilizzate**

* **Keycloak**: Server IAM (Identity and Access Management).
* **Docker**: Esecuzione di Keycloak in container locale.
* **Python + Flask**: Sviluppo API REST (Libera e Protetta).
* **python-keycloak, PyJWT, requests**: Librerie Python per gestione Token e chiamate API.

**3. Architettura del Progetto**

* **Keycloak Realm**: tesi
* **Client in Keycloak**: python-client (Public)
* **Utente Keycloak**: studente / password: 1234
* **API Libera**: Accessibile senza autenticazione (porta **5001**)
* **API Protetta**: Richiede Token JWT per accesso (porta **5002**)

**4. Setup Ambiente**

**4.1 Avvio Keycloak con Docker**

docker compose up

**4.2 Configurazione Keycloak**

1. Crea Realm: tesi
2. Crea Client: python-client (Access Type: Public)
3. Crea Utente: studente / password: 1234

**4.3 Creazione Ambiente Virtuale Python**

python3 -m venv venv

source venv/bin/activate

python3 -m pip install python-keycloak Flask PyJWT requests

**5. File del Progetto**

**5.1 api\_libera.py**

API libera accessibile senza token (porta **5001**).

**5.2 api.py**

API protetta che richiede Token JWT valido (porta **5002**).

* Verifica la firma del token con le chiavi pubbliche JWKS di Keycloak.
* Se il token è mancante → **403 Forbidden**.
* Se il token è invalido o scaduto → **401 Unauthorized**.
* Se il token è valido → **200 OK**.

**5.3 client.py**

Effettua richiesta di autenticazione a Keycloak e restituisce il Token JWT ottenuto.

**5.4 call\_api.py**

Script di test completo che:

* Richiede Token JWT.
* Esegue chiamata alla API libera.
* Esegue chiamata alla API protetta senza token (atteso fallimento).
* Esegue chiamata alla API protetta con token (dimostrazione protezione attiva).

**6. Esecuzione del Progetto**

**Terminale 1: API Protetta**

source venv/bin/activate

python3 api.py

**Terminale 2: API Libera**

source venv/bin/activate

python3 api\_libera.py

**Terminale 3: Esecuzione Test**

source venv/bin/activate

python3 call\_api.py

**7. Output della Dimostrazione**

* **Token JWT generato da Keycloak** (stampato a video, ad esempio: eyJhbGciOiJSUzI1NiIsInR5cCIg...).
* **Chiamata API Libera → 200 OK**
* {"message": "Connessione riuscita alla API libera (senza autenticazione)"}
* **Chiamata API Protetta senza token → 403 Forbidden**
* {"error": "Accesso vietato: token mancante"}
* **Chiamata API Protetta con token → 200 OK**
* {
* "message": "✅ Accesso consentito con token valido",
* "subject": "af89dbdf-c08c-4d0f-93b1-94459b316c78",
* "issued\_at": 1756115464,
* "expires\_at": 1756115764
* }

**8. Considerazioni**

* Il progetto dimostra la protezione delle API tramite meccanismi di autenticazione basati su Token (OAuth 2.0).
* La gestione dei ruoli (role mapping e policies) non è stata implementata per semplicità.
* Il token viene validato in firma (JWT Signature Verification).

**9. Possibili Sviluppi Futuri**

* Implementazione Role-Based Access Control (RBAC) con Keycloak.
* Gestione di scopes e policies avanzate (Authorization Services).
* Deployment in ambienti Cloud o containerizzati (Kubernetes).

**10. Screenshot da Produrre per la Tesi**

* Console Keycloak (Realm, Client, Utente).
* Token JWT generato da Keycloak.
* Chiamata API libera (200 OK).
* Chiamata API protetta senza token (403 Forbidden).
* Chiamata API protetta con token (200 OK).

Vuoi che ti generi direttamente il **file DOCX impaginato** con questa guida aggiornata, così lo puoi allegare subito alla tesi