ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

INSTALLAZIONE DI UN IDM KEYCLOAK PER L'AUTENTICAZIONE DEGLI UTENTI E CONFIGURAZIONE DI UN SSO TRA DIVERSE APPLICAZIONI: UN CASO DI STUDIO AZIENDALE.

Elaborato in
TECNOLOGIE WEB

Relatore
Prof.ssa SILVIA MIRRI

Presentata da
SARA RAGNETTI

Anno Accademico 2021 – 2022

A me che, come ogni Nesta là fuori,
ho scalato la mia montagna.

Indice

Introduzione

La gestione delle identità e il controllo accessi sono da sempre due punti centrali della sicurezza informatica: si tratta, infatti, di misure di sicurezza previste dai framework di cyber security e di sicurezza informatica. Con il rapido aumento delle diverse tipologie di applicazioni e sistemi aziendali on premise e in cloud, unito al fatto che ciascun utente dispone di svariati account e di più privilegi per ogni account, si è palesata la necessità di avere un controllo degli accessi efficace incentrato non tanto sugli account stessi, quanto sull'identità, in modo che fosse possibile visualizzare e tracciare con precisione "chi ha fatto cosa" e valutare che chi ha accesso ad una qualche risorsa ne abbia, di fatto, il diritto.

In un sistema informatico aziendale in cui è necessario poter accedere a qualsiasi risorsa da qualsiasi luogo, è cruciale basare sia i diritti di accesso alle risorse che le decisioni sui rischi correlati a tali accessi sull'identità. Purtroppo però, sebbene i sistemi di gestione dell'accesso e dell'identità siano disponibili già da tempo, nella maggior parte dei casi solo le aziende medio grandi ne sono dotate, e spesso non sono utilizzati al meglio delle loro possibilità.

La maggior parte delle organizzazioni, infatti, si limita ancora alla gestione degli account piuttosto che delle identità, venendo meno al privilegio di cogliere l'importante opportunità di sviluppare un programma evolutivo attraverso cui rafforzare la sicurezza delle informazioni aziendali. L'obiettivo principale dei sistemi IAM è identificare un utente, autenticarlo e autorizzane l'accesso in base al suo profilo e al ruolo dichiarato nella sua identità digitale. Una volta stabilita, l'identità digitale deve essere mantenuta, modificata e monitorata in ogni momento del ciclo di vita del procedimento di accesso e cancellata al momento appropriato.

Imola Informatica è infatti una grande azienda che si serve di più applicazioni e servizi. Questo porta i dipendenti a dover accedere a più app con set di credenziali diverse. In questo contesto è nata l'idea, all'interno di Imola Informatica, di adottare una soluzione di Single sign-on in modo da limitare al massimo la proliferazione di account in rete, e di gestire gli accessi alle risorse attribuendo agli utenti dei permessi diversi in base al ruolo, centralizzando, di fatto, la gestione delle utenze.

Tra gli obiettivi di questo elaborato vi è innanzitutto l'acquisizione delle conoscenze relative ai sistemi per la gestione delle identità e degli accessi necessaria per comprenderne al meglio l'importanza. Poi, la configurazione di un Identity Manager Keycloak per l'autenticazione degli utenti e l'implementazione di due API, una per ciascuna applicazione realizzata. Infine, la configurazione di un sistema di single sign-on tra le due applicazioni mediante l'acquisizione e la decodifica di un token scambiato tra le parti.

La tesi è strutturata come segue:

- **capitolo 1**: nel primo capitolo viene introdotto il lavoro di ricerca effettuato sullo stato dell'arte, analizzando le motivazioni che spingono le aziende a tutelare con sempre più attenzione le identità dei propri dipendenti ed approfondendo alcuni aspetti legati alle identità digitali;

- **capitolo 2**: nel capitolo 2 viene contestualizzato l'argomento di studio all'interno dell'azienda Imola Informatica. Si parla anche del progetto di tesi, esplicitando i motivi per cui sono state scelte determinate tecnologie;

- **capitolo 3**: infine, in quest'ultimo capitolo, vengono elencati, innanzitutto, gli strumenti e i framework utilizzati, poi si descrivono tutti gli step per l'istallazione di un IDM Keycloak per l'autenticazione degli utenti e per l'instaurazione di un social login mediante GitHub. Vengono, inoltre, analizzate nel dettaglio le implementazioni delle API delle due applicazioni e i loro casi d'uso.

Capitolo 1

IAM (Identity and Access Management)

Nei prossimi paragrafi si vedrà nel dettaglio cos'è un sistema IAM, quali tecniche vengono adottate per realizzarlo e soprattutto la misura in cui esso è rilevante per gli accessi ad una risorsa informatica all'interno di una organizzazione.

Essendo l'identità un concetto estreamamente importante nel contesto di questa analisi, vi sarà dedicato un intero paragrafo, con particolare attenzione a quella che è l'identità coinvolta nei sistemi IAM, ovvero l'identità digitale.

Dopo aver delineato le differenze tra sistemi IAM centralizzati e decentralizzati, mettendo in luce punti di forza e debolezze di entrambi, si porrà l'attenzione sui sistemi del primo tipo e sul loro più classico esempio di applicazione: il single sign-on.

1.1 Cos'è e come funziona un sistema IAM

Il problema del riconoscimento dell'utente e dell'attribuzione dei permessi a lui associati è un problema che si verifica laddove vengono fornite particolari risorse informatiche (risorse web, accesso a Internet, posta elettronica, ecc.) che richiedono fasi di autenticazione e di autorizzazione.

Figura 1.1: Fasi del processo di accesso alle risorse.

E' qui che entra in gioco l'Identity and Access Management (IAM, in breve), un sistema ideato per gestire le informazioni riguardanti le identità degli utenti e a controllarne l'accesso alle risorse informatiche.

La gestione delle identità e degli accessi è anche, in un certo senso, un insieme di regole generali che permette di svolgere le seguenti operazioni:

- autenticazione, ovvero assicurare che gli utenti dimostrino di essere chi dichiarano di essere;
- autorizzazione, ovvero assicurare che gli utenti, dopo essere stati autenticati, abbiano accesso solo ed esclusivamente alle risorse a cui hanno diritto. In altre parole, indica chi è autorizzato a eseguire determinate operazioni sulla base di una comprovata identità;
- amministrazione, ovvero avere la capacità di gestire il ciclo di vita dell'identità e gli accessi di tutti gli utenti secondo le politiche aziendali;
- analisi, ovvero avere la capacità di rilevare gli accessi impropri;
- audit, ovvero verificare la correttezza delle autorizzazioni assegnate e il rispetto delle policy e delle procedure aziendali mediante il tracciamento degli accessi.

I sistemi IAM sono formati dall'unione di due componenti distinti che devono poter comunicare e lavorare in stretta relazione: i sistemi di Identity Management (IdM) e di Access Management (AM).

1.1.1 Identity Management (IdM)

L'Identity Management (IdM) è l'insieme dei processi che permettono la creazione, il mantenimento e l'uso delle identità digitali in un contesto legale di tipo amministrativo, commerciale, pubblico o privato.

Nel contesto di un IT aziendale, l'IdM è il processo per definire e gestire i ruoli e i privilegi di accesso dei singoli utenti della rete secondo le esigenze di lavoro, se l'utente è un dipendente e secondo le condizioni contrattuali, se si tratta invece di un cliente.

I sistemi di Identity Management permettono di gestire e automatizzare tutti gli account aziendali che fanno riferimento ad una stessa identità, attraverso il provisioning (assegnazione dell'utilizzo di certe risorse all'utente) e deprovisioning (revoca dell'utilizzo di certe risorse all'utente) automatico degli account al momento di inizio e fine della collaborazione, del cambio di mansioni, basandosi sul ruolo dell'utente.

Un'altra funzione essenziale degli IM è quello di fornire strumenti che consentano agli utenti di risolvere determinati problemi autonomamente (self-service), senza che siano richieste particolari competenze informatiche. L'esempio più tipico sono gli strumenti per il reset della password.

1.1.2 Access Management (AM)

I sistemi di Access Management (AM) si occupano dell'autenticazione centralizzata sui sistemi e sulle applicazioni aziendali e dell'autenticazione federata, agendo sia come Identity Provider permettendo quindi agli utenti di utilizzare le credenziali aziendali per accedere ad applicazioni fornite da terzi, sia come Service Provider consentendo ad eventuali utenti esterni autorizzati di utilizzare le credenziali delle loro aziende per accedere ad applicazioni aziendali. I sistemi AM implementano il Single-Sign-On e il controllo dell'accesso, e possono utilizzare diversi sistemi di autenticazione e diversi livelli di verifica dell'identità (Multi Factor Authentication) in base al rischio che deriva dal tipo di accesso o in relazione alla criticità della risorsa per cui viene richiesto l'accesso. Forniscono inoltre API, web service, interfacce SAML e Oauth che permettono l'integrazione con diverse tipologie di applicazioni e altri sistemi di autenticazione. Infine, alcuni sistemi AM svolgono il ruolo di Mobile Application Management (MAM), ovvero permettono l'esposizione su Internet di applicazioni mobile aziendali basate su tecnologie Web, implementando un meccanismo che permetta la separazione dei dati aziendali utilizzati all'interno del dispositivo mobile dai dati privati degli utenti.

Contestualmente al controllo degli accessi bisogna prestare particolare attenzione alla gestione degli account "privilegiati" che, a causa degli ampi privilegi di accesso loro assegnati, possono accedere ad informazioni critiche anche senza averne effettivamente la necessità ai fini dello svolgimento del proprio lavoro.

1.2 Perché efettuare il controllo degli accessi: la "triade della CIA"

Il motivo principale per cui le organizzazioni implementano dei meccanismi di controllo degli accessi è per evitare perdite di informazioni e risorse. Tali perdite si suddividono in: perdita di riservatezza, perdita di disponibilità e perdita di integrità. In materia di sicurezza IT, l'insieme delle tre viene chiamato "la triade della CIA" (o triade della sicurezza).

Riservatezza (confidentiality) La perdita di riservatezza si verifica quando delle entità accedono al sistema o ai dati senza averne l'autorizzazione. I controlli di accesso filtrano i soli soggetti autorizzati.

Integrità (integrity) L'integrità garantisce che non vengano apportate modifiche ai dati o alle configurazioni di sistema senza autorizzazione. Se viene effettuata una modifica non autorizzata, i controlli di sicurezza la rilevano immediatamente per mantenere la riservatezza. Se, malauguratamente, si verificassero delle modifiche non pertinenti o non autorizzate, si avrebbe una perdita di integrità.

Disponibilità (availability) I soggetti autorizzati devono essere abilitati a cercare le risorse di cui necessitano entro una quantità di tempo ragionevole. E' importante che dati e sistemi siano disponibili ogni qual volta che un utente o un altro soggetto ne abbia bisogno. Se i sistemi non riescono a fornire informazioni valide in tempo, allora risulteranno essere non operativi e i dati saranno considerati inaccessibili.

1.3 Identità e ciclo di vita dell'identità

Diamo ora una definizione dettagliata di identità e analizziamo il suo ciclo di vita.

L'identità è un termine a cui sono associati molti significati. In senso filosofico o psicologico, il concetto di identità riguarda la concezione che un individuo ha di se stesso nell'individuale e nella società. In altre parole, è l'insieme delle caratteristiche che lo rendono unico e inconfondibile, cioè distinguibile da altri.

In contesti diversi, descrive la condizione di essere se stessi (e non altri) sulla base di attributi identificativi personali statici, cioè che difficilmente cambieranno nel tempo.

E' proprio in questo contesto che il concetto di "verifica dell'identità" si è evoluto. L'identificazione è la prova che può essere fornita per dimostrare che una presunta identità è autentica e quindi, per autenticare quell'identità non solo nella realtà, ma anche nel digitale.

Purtroppo i termini "identità digitale", "footprint digitale" e "identificazione elettronica" sono usati in modo intercambiabile, anche se si tratta di concetti molto diversi.

Per chiarire la terminologia, ecco le definizioni riportate nell'articolo del dipartimento di sicurezza informatica dell'Università di Ulster [1].

Identità digitale L'insieme di informazioni che, all'interno di un determinato sistema informatico, si riferiscono a una specifica persona prende il nome di identità digitale. Ciascuno può avere più identità digitali associate a diversi account: e-mail, social media, ecc. Questo implica che un servizio non è in grado di identificare univocamente un'identità in tutti i contesti o, in altre parole, che una stessa identità digitale possa riferirsi a soggetti reali diversi, in contesti differenti.

Per questo motivo, un'identità digitale potrebbe essere associata o meno all'identità reale del soggetto, così come può essere associata a varie organizzazioni attraverso record elettronici, sistemi IAM, firme digitali e certificati.

In tal senso, il processo di verifica dell'identità serve proprio a stabilire se il soggetto che tenta di effettuate l'accesso presso un servizio digitale sia chi afferma di essere.

Footprint digitale Ogni volta che si utilizza Internet, si lascia una scia di informazioni che prende il nome di footprint digitale. Le dimensioni del footprint crescono man mano che si svolgono determinate azioni sulla Rete.

Non è sempre evidente che si sta contribuendo ad alimentare il proprio footprint digitale ed è proprio in questo che sta la differenza tra impronta "attiva" e "passiva".

Un footprint digitale è di tipo attivo se l'utente ha deliberatamente condiviso informazioni personali, ad esempio pubblicando contenuti oppure interagendo su social network o forum online. Se un utente ha effettuato l'accesso a un sito Web con un nome utente o con un profilo registrato, tutti i post pubblicati fanno parte del suo footprint digitale attivo.

Altri esempi di attività che contribuiscono ai footprint digitali attivi sono la compilazione di un modulo online, l'iscrizione a una newsletter, o il consenso ad accettare i cookie nel browser.

Un footprint digitale passivo, invece, viene creato quando vengono raccolte informazioni sull'utente senza che lui lo sappia o se ne renda conto, come avviene, ad esempio, quando i siti Web raccolgono informazioni sul numero di visite effettuate dagli utenti, sulla loro provenienza e sul loro indirizzo IP.

Un altro esempio di footprint passivo è l'utilizzo che i siti di social network e gli inserzionisti fanno di like, condivisioni e commenti per profilare gli utenti e indirizzargli contenuti specifici.

Identificazione elettronica Questo tipo di identificazione, sebbene digitalizzata, si riferisce in gran parte a registri elettronici che contengono dati relativi a cittadini di una nazione. Oltre a consentire l'accesso online e l'autenticazione per i servizi governativi, questo record digitale è collegato a un

documento fisico contenente un chip che ne verifica l'identità del titolare del documento.

Per questo motivo, l'identificazione elettronica è generalmente considerata un'autenticazione a due fattori, in cui un soggetto "possiede" qualcosa e "conosce" una password per autenticare la propria identità.

1.3.1 Identità digitale

In questa sede si farà riferiemento all'identità digitale intesa in quanto tale. Analizziamola ora nel dettaglio.

L'identità digitale è costituita da:

- *dati anagrafici*: informazioni che descrivono un'entità e la identificano rispetto alla realtà. Questi dati possono subire cambiamenti, ma indipendentemente dai cambiamenti dell'entità rispetto ad una organizzazione. Dati anagrafici possono essere nome, cognome, luogo e data di nascita.

- *attributi*: informazioni dell'entità nel contesto di una organizzazione. Si tratta di ruoli istituzionali, privilegi, incarichi, appartenenza a gruppi.

- *credenziali*: username e password usate per accedere alle risorse informatiche. Sono mutabili, possono cioè cambiare nel tempo per volontà dell'utente o in casi eccezionali, come il caso in cui l'utente non si ricorda la chiave di accesso.

Come anticipato in precedenza, un utente potrebbe avere più di un'identità digitale, una per ciascun account ad esso associato. Questa non univocità implica, da parte dell'utente, dover memorizzare più credenziali e, da parte del sistema, dover gestire un'eccessiva mole di informazioni, spesso e volentieri in contrasto fra loro.

Avere più identità digitali per un'unica entità comporta una gestione complessa e confusionaria delle informazioni, in termini soprattutto di coerenza dei dati. Sarebbe infatti opportuno che le modifiche apportate ad una identità venissero propagate anche a tutte le altre.

Diviene quindi fondamentale trovare un insieme, un gruppo di attributi da associare agli utenti in modo da identificare senza ambiguità ognuno di essi. Sarà poi opportuno riferirsi a loro attraverso i suddetti attributi, che prendono il nome di chiave. Si creerà in questo modo un'identità digitale unica per ogni utente.

1.3.2 Ciclo di vita

Molte organizzazioni IT e di sicurezza aziendali si affidano a processi manuali inefficienti per il provisioning di nuovi utenti e la gestione dei privilegi ad essi associati. Talvolta, in casi estremi, sono necessarie addirittura settimane per accogliere i nuovi assunti e fornire loro un accesso sicuro alle applicazioni e ai sistemi IT di cui hanno bisogno per svolgere efficacemente il proprio lavoro.

In più, va aggiunto che molte aziende non dispongono di processi formali o automatizzati per il ripristino dei privilegi o la disattivazione degli account utente quando i lavoratori cambiano ruolo o mansione o lasciano l'azienda. Come conseguenza, i conti rimangono attivi anche dopo che i dipendenti non lavorano più presso l'azienda o hanno cambiato posizione. I lavoratori scontenti, gli appaltatori non autorizzati e gli avversari possono sfruttare questi account dormienti (ancora aperti) o i privilegi di utente obsoleti per lanciare attacchi o rubare dati riservati.

Le soluzioni di gestione del ciclo di vita delle identità pongono rimedio a queste problematiche, automatizzando i processi di provisioning degli utenti e di governance delle identità, eccessivamente onerosi da svolgere manualmente e fin troppo soggetti ad errori dovuti all'intervento umano/manuale. Aumentano la produttività dei dipendenti consentendo ai nuovi assunti di iniziare a lavorare fin da subito con l'accesso immediato ad applicazioni e servizi IT e aiutano le aziende a ridurre i rischi per la sicurezza, eliminando gli account utente scaduti insieme ai privilegi ad essi associati. Inoltre, aiutano le organizzazioni con il proprio business distogliendo il personale da queste attività affinché si concentri su altre più importanti.

Le best practice per la gestione del ciclo di vita dell'identità comprendono diverse fasi.

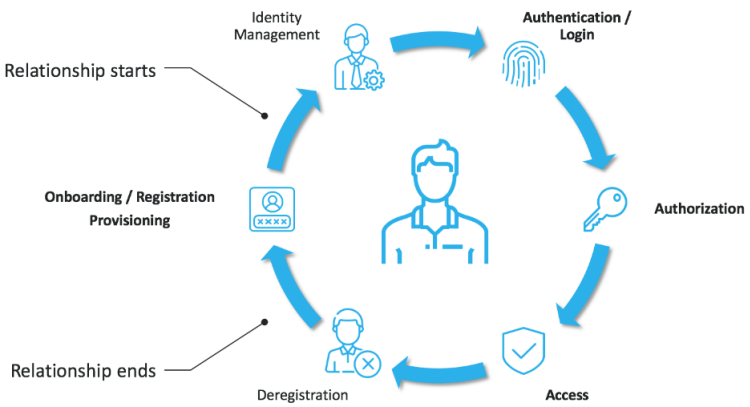


Figura 1.2: Schema del ciclo di vita dell'identità [2].

Provisioning

La prima fase della gestione del ciclo di vita dell'identità è la creazione dell'identità stessa. Questo di solito avviene poco prima o comunque il primo giorno di lavoro di un nuovo dipendente e se ne occupa il reparto di amministrazione o delle risorse umane. Le risorse umane raccolgono le informazioni rilevanti dal neo-assunto, le utilizzano per creare la propria identità digitale a cui associano un ruolo o un titolo e delle credenziali con cui avranno accesso a risorse, sistemi o servizi in base a ruoli e privilegi.

Onboarding

Una volta che il team HR ha creato l'identità digitale del nuovo dipendente e qusta diventa attiva nel software delle risorse umane, deve essere creata anche all'interno del software IT.

Una volta importato, il ruolo associato all'identità indica al software IT quale accesso deve essere fornito. Questo viene spesso fatto tramite i *gruppi*. Quindi, quando una nuova identità viene associata ad un gruppo, le viene immediatamente assegnato anche l'accesso a determinate risorse associate al gruppo stesso.

Aggiornamento/modifiche

Questa fase prevede alcune attività diverse, ma correlate. Prima fra tutte, il monitoraggio dei livelli di accesso per garantire che nessun account sia scaduto o non privilegiato con il passare del tempo. E' prevista, inoltre, la redazione di report in merito a identità e accessi per dimostrare che la conformità è stata raggiunta e che gli standard di sicurezza sono stati rispettati. Infine, in questa terza fase è garantita la manutenzione continua degli accessi durante i cambi di ruolo o di responsabilità.

Deprovisioning

La fase finale del ciclo di vita dell'identità è l'offboarding. Un corretto offboarding è essenziale per mantenere la sicurezza all'interno dell'organizzazione, perché gli ex dipendenti che mantengono l'accesso a una qualsiasi risorsa rappresentano una minaccia. Per far fronte a questo problema, è fondamentale revocare l'accesso e deprovisionare le identità in modo tempestivo, completo e sicuro.

1.4 I processi di identificazione e autenticazione

L'identificazione è il processo attraverso il quale un soggetto rivendica un'identità. Questo si verifica ogni qual volta l'utente deve fornire la propria identità ad un sistema durante i processi di autenticazione, autorizzazione e rendicontabilità. Può avvenire in diversi modi: digitando il nome utente, dicendo una frase o posizionando la mano, il dito o il viso davanti ad una telecamera o davanti ad un dispositivo di scansione.

E' importante che tutti i soggetti abbiano identità univoche. L'autenticazione confronta uno o più fattori con un database di identità valide, come lo user account, per verificare l'identità.

Le informazioni che gli utenti utilizzano per verificare la propria identità sono private e devono essere protette. Ad esempio, le password non vanno assolutamente salvate in chiaro nel database.

I sistemi di autenticazione, infatti, memorizzano gli hash delle password. Il livello di sicurezza di un sistema di autenticazione dipende principalmente dalla capacità sua e da quella dei soggetti di mantenere la segretezza delle informazioni. Identificazione ed autenticazione avvengono sempre insieme come se fossero un'unica fase dello stesso processo divisa in due step.

Dichiarare la propria identità (primo step) e fornire le informazioni per l'autenticazione (secondo step) sono i passaggi da svolgere necessariamente per accedere ad un sistema. La combinazione di un nome utente e di una password è un semplice esempio di identificazione e processo di autenticazione. Per identificarsi, gli utenti utilizzano i loro nomi utente, e per autenticarsi, forniscono le password.

1.4.1 Registrazione e verifica dell'identità

Per ottenere un'identità per la prima volta, l'utente deve eseguire un processo di registrazione. All'interno di un'organizzazione, i nuovi dipendenti, al momento dell'assunzione, devono fornire dei documenti per dimostrare la loro identità. Successivamente, i dipendenti del dipartimento delle Risorse Umane (HR) iniziano il processo di creazione degli ID utente.

La verifica dell'identità viene utilizzata quando gli utenti interagiscono, ad esempio, con siti online. Pensiamo ad un sito di online banking; durante la creazione di un account, la banca prenderà alcune misure per convalidare l'identità dell'utente. Di solito, chiederanno informazioni note solo all'utente e alla banca, come i numeri di conto e informazioni personali sull'utente. Inoltre, durante il processo di registrazione iniziale, la banca chiederà all'utente di fornire alcune informazioni aggiuntive come il nome del primo animale domestico,

il colore preferito dell'utente o il modello della sua prima auto. In seguito, se l'utente desidera, ad esempio, modificare la propria password o trasferire del denaro, la banca può porgere quelle stesse domande per verificarne l'identità.

La verifica dell'identità non è altro che il procedimento che conferma che l'utente è realmente chi dice di essere. Potrebbe sembrare un'autenticazione classica, basata su una combinazione di nome utente/password, ma il controllo dell'identità entra effettivamente in gioco prima che gli utenti ottengano le proprie credenziali per accedere ad un'applicazione.

L'obiettivo finale del controllo dell'identità è garantire che l'identità rivendicata da un utente corrisponda alla sua identità effettiva, cioè che la sua identità sia reale e non fittizia. Per capire meglio questo concetto prendiamo in considerazione due definizioni date dal National Institute of Standards and Technology (NIST) nelle sue Linee guida [3] per l'identità digitale.

L'identità rivendicata si riferisce ai dati sull'identità dichiarata da un utente quando si registra con un sistema IAM, ovvero chi afferma di essere. L'identità effettiva invece è relativa ai dati che dimostrano l'autenticità dell'identità di un utente o, in altre parole, chi è realmente.

Il giusto sistema di verifica dell'identità può eseguire il proprio lavoro automaticamente: raccogliere informazioni sull'utente, verificare se la sua identità dichiarata corrisponde alla sua identità effettiva e approvare la sua registrazione o richiesta di accesso. Tutto questo avviene in tempo reale, senza intervento umano, creando una funzione di autenticazione sicura. Investire in servizi di verifica dell'identità evita di far sprecare risorse uomo, cioè di distogliere alcuni dipendenti dalle loro competenze principali.

1.4.2 Autorizzazione e rendicontabilità

Autorizzazione e rendicontabilità sono due elementi di sicurezza aggiuntivi in un sistema di controllo di accesso.

L'autorizzazione è il processo di concessione dell'accesso a soggetti sulla base di identità verificate. Assicura che l'attività richiesta o l'accesso alla risorsa è possibile in base ai privilegi assegnati, indicando se l'utente è autorizzato o meno a eseguire determinate operazioni. Se l'azione è consentita, il soggetto è autorizzato; in caso contrario, non è autorizzato.

Il fatto che l'utente possa autenticarsi in un sistema non significa che abbia, ad esempio, il permesso di aprire e sovrascrivere qualsiasi file. Se un dipendente di un'organizzazione tenta di aprire un file, il sistema di autorizzazione controlla se quel dipendente ha perlomeno il permesso di leggerlo.

L'auditing è il processo di monitoraggio e registrazione delle attività degli utenti attraverso dei file log. Questi file, in genere, registrano chi ha svolto

una determinata azione, qual è stata questa azione, e quando e dove è stata intrapresa.

Uno o più log creano una traccia utile per ricostruire gli eventi e identificare eventuali falle nella sicurezza. In questo modo, quando gli investigatori esaminano il contenuto delle tracce, saranno in grado di risalire ai responsabili e fornire prove necessarie.

1.4.3 Fattori di autenticazione

I fattori di autenticazione sono delle informazioni che vengono usate per verificare l'identità e l'autorizzazione di un utente che tenta di ottenere l'accesso o di inviare richieste di dati ad una rete sicura o ad un'applicazione. Generalmente le organizzazioni fanno affidamento su username univoci e password selezionate dall'utente come principale metodo di autenticazione. Ma ora, come conseguenza dell'aumento dell'attenzione nei confronti dei controlli di sicurezza a causa della regolamentazione, molti gruppi aziendali utilizzano più fattori di autenticazione per controllare l'accesso a sistemi e applicazioni protetti.

Esistono tre metodi di autenticazione principali, i cosiddetti fattori di autenticazione:

- qualcosa che conosci (something you know): come una password, un ID personale, un codice numerico (PIN) o una passphrase;
- qualcosa che hai (something you have): gli esempi includono qualsiasi dispositivo fisico che possiede l'utente e che possa quindi aiutarlo con l'autenticazione;
- qualcosa che sei (something you are): ovvero una o più caratteristiche fisiche della persona, dati biometrici come impronte digitali e vocali, forma del viso, tipologia del palmo, ma anche dati di biometria comportamentale.

Se implementato correttamente, un fattore di autenticazione di tipo "something you are" è il più robusto, mentre uno di tipo "something you know" è il più debole. Quindi una password è più debole di un'impronta digitale. In ogni caso, ci sono diversi modi che i malintenzionati possono usare per aggirare persino il più robusto dei fattori di autenticazione.

1.4.4 Authenticazione a più fattori

L'autenticazione a più fattori è un processo di autenticazione che prevede l'utilizzo di due o più fattori. Il primo esempio che può venire in mente è

quando si deve prelevare denaro da un bancomat; non basta inserire la propria carta per effettuare l'operazione, ma verrà anche richiesta anche la digitazione di un pin.

Un'autenticazione di questo tipo, in linea di massima, è più sicura, ma occorre prestare attenzione a come viene implementata. Se viene chiesto agli utenti di inserire una password e un PIN, infatti, non si è di fronte ad un'autenticazione a più fattori poiché entrambi i metodi sono della stessa tipologia di fattore (something you know). Usare più metodi dello stesso fattore non è più sicuro rispetto ad usarne uno solo perché lo stesso attacco che potrebbe rubare l'uno potrebbe procurarsi anche l'altro.

1.5 Tecniche per la gestione delle identità

Le tecniche di gestione dell'identità possono essere suddivise in due categorie:

- centralizzate;
- decentralizzate.

1.5.1 IAM centralizzato

La gestione centralizzata delle identità e degli accessi è una tecnica per l'archiviazione e la gestione dei dati degli utenti in un'unica posizione. Fornisce un processo sicuro per identificare, autenticare e autorizzare gli utenti che hanno il permesso di accedere alle risorse digitali di un'azienda.

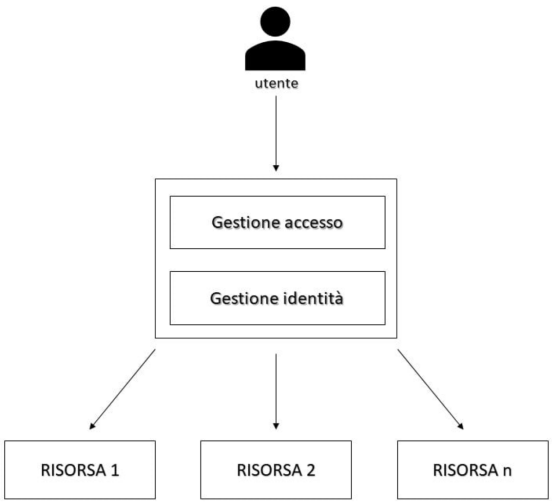


Figura 1.3: Schema d'esempio di uno IAM centralizzato.

Con l'IAM centralizzato, gli utenti possono accedere a tutte le risorse e le applicazioni di cui hanno bisogno per svolgere il proprio lavoro, inserendo un solo set di credenziali di accesso. L'eliminazione della necessità di ricordare e mantenere ID e password di accesso separati per ciascuna risorsa e la riduzione al minimo del numero di reimpostazioni delle password migliorano l'esperienza complessiva dell'utente.

Il componente di identità dell'IAM centralizzato consolida l'archiviazione e la gestione dei dati relativi all'identità, incluse le credenziali di accesso, i ruoli e le autorizzazioni di ciascun utente. L'archiviazione di queste informazioni in un repository centrale semplifica il provisioning e il deprovisioning degli utenti rendendoli automatici e offre ai team IT la possibilità di osservare l'attività di accesso degli utenti per tutte le risorse, indipendentemente dalla posizione. Con una maggiore visibilità, i team possono rilevare le minacce più velocemente e impedirne la diffusione.

Un esempio pratico

La violazione della sicurezza di Uber dello scorso settembre sottolinea la necessità di misure di sicurezza più forti come l'IAM centralizzato per impedire agli hacker di utilizzare credenziali rubate per ottenere l'accesso alle risorse aziendali e ai dati sensibili [4].

In breve, un malintenzionato ha compromesso l'account di un appaltatore Uber EX, probabilmente acquisendo la password aziendale Uber sul dark web, dopo che il dispositivo dell'appaltatore era stato infettato da malware, esponendo tali credenziali. L'aggressore ha tentato ripetutamente di accedere all'account Uber dell'appaltatore che ad ogni nuovo tentativo riceveva una richiesta di approvazione dell'autenticazione a due fattori, cosa che inizialmente bloccava l'accesso. Alla fine l'appaltatore ha accettato una delle tante richieste e il malintenzionato ha effettuato l'accesso con successo. In questo modo l'attaccante ha avuto accesso a diversi altri account dei dipendenti che alla fine gli hanno concesso autorizzazioni per una serie di strumenti, tra cui G-Suite e Slack. ll malintenzionato ha sfruttato le informazioni e le risorse ottenute per pubblicare un messaggio su un canale Slack a livello aziendale e per riconfigurare gli OpenDNS di Uber. Il 19 settembre dello scorso anno, lo stesso Uber ha affermato che i propri processi di monitoraggio di sicurezza hanno consentito ai team dedicati di identificare rapidamente il problema e di reagire con la priorità di assicurarsi che l'aggressore non avesse più accesso ai sistemi per garantire che i dati degli utenti fossero al sicuro e che i servizi Uber non fossero interessati. Con uno IAM centralizzato, un attacco di questo tipo è stato più facile da individuare e isolare grazie alla concentrazione dei controlli di sicurezza in un unico punto. Vediamo come proprio questa concentrazione

dei meccanismi di sicurezza possa essere anche una debolezza dei sistemi IAM centralizzati.

Sfide della gestione centralizzata delle identità

Sebbene l'IAM centralizzato rafforzi la sicurezza fornendo controlli più rigorosi che aiutano a prevenire l'accesso non autorizzato, non è una strategia perfetta. I critici di un approccio centralizzato sostengono che la singola archiviazione di identità sia il problema più preoccupante. Affidarsi a un solo set di credenziali crea infatti un single point of failure. Hackerando con successo l'account di un utente, un malintenzionato potrebbe ottenere l'accesso a tutte le risorse per cui l'utente ha i privilegi di accedere. Tuttavia, si può porre rimedio a questo inconveniente. Le organizzazioni possono infatti mitigare il rischio implementando protocolli di autenticazione più forti, come l'autenticazione a più fattori (MFA) o la biometria.

1.5.2 IAM decentralizzato

Con la gestione decentralizzata delle identità, nota anche come gestione dell'identità distribuita, l'accesso è distribuito su più ambienti. Questo implica che gli utenti debbano accedere alle applicazioni individualmente, utilizzando diversi set di credenziali per ciascuna. Il modello decentralizzato distribuisce le identità degli utenti attraverso la rete, poiché ogni applicazione deve archiviare e gestire i propri dati utente, eliminando di fatto il single point of failure introdotto da un sistema IAM centralizzato mal implementato.

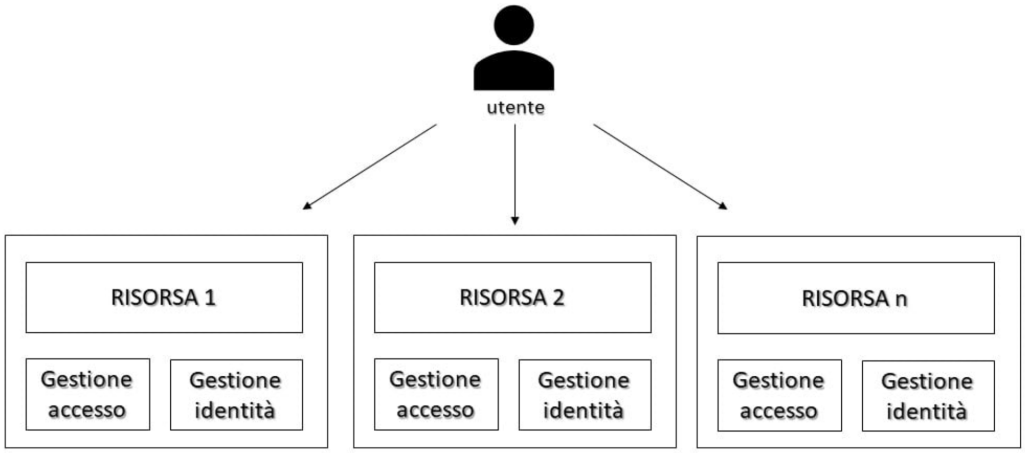


Figura 1.4: Schema d'esempio di uno IAM decentralizzato.

Una gestione decentralizzata delle identità offre, da un lato, un maggiore controllo agli utenti e dall'atro, meno visibilità alle aziende. Nello specifico, tale tecnica permette di archiviare i dati relativi all'identità in un portafoglio digitale sul proprio dispositivo mobile dove verrà creata anche una coppia di chiavi pubbliche e private che consentiranno all'utente di condividere solo le informazioni necessarie per completare un'operazione. In questo modo sarà l'utente ad avere sempre il controllo, e non l'organizzazione e potrà quindi mantenere aggiornate tutte le informazioni e le proprie preferenze di condivisione. La capacità di possedere e controllare i propri dati privati è uno dei principi fondamentali dell'identità auto-sovrana, ampiamente diffusi nella tecnologia blockchain.

Sfide della gestione decentralizzata delle identità

Avere più squadre di persone che lavorano su diversi punti di controllo degli accessi rende difficile implementare forti meccanismi di sicurezza e, in generale, il mantenimento del sistema e fa aumentare le spese amministrative. Le informazioni sulle identità sono infatti replicate per ogni risorsa e l'aggiunta di una nuova risorsa implica l'aggiunta di un'intera infrastruttura di accesso. Per quanto riguarda gli utenti, invece, il fatto che ognuno si ritrovi ad avere credenziali di accesso diverse per ogni applicazione che utilizza rende più probabile la scelta di password semplici da parte degli utenti, oppure l'annotazione delle stesse su foglietti di carta o blocco note. Questo implica palesemente una minore sicurezza per i dati. Le tecnologie decentralizzate non possono, infine, eguagliare il controllo amministrativo che l'IAM centralizzato offre alle organizzazioni. Sacrificando la visibilità sulle attività degli utenti e sulle risorse dei sistemi, il rischio di una violazione aumenta perché le minacce sono più difficili da rilevare e isolare.

1.6 Single Sign-On (SSO)

1.6.1 Che cos'è il single sign-on

Il single sign-on (SSO) è l'esempio principale di tecnica di controllo degli accessi centralizzata. Consente a un soggetto di accedere a più risorse su un sistema ottenendo l'autenticazione una sola volta. Per esempio, se un utente vuole connettersi ad una rete e il sistema gli garantisce l'accesso in seguito all'autenticazione, sarà in grado di accedere alle risorse in tutta la rete senza che venga richiesto di autenticarsi nuovamente.

Il SSO non è solo facile da usare, ma aumenta anche la sicurezza. Quando gli utenti devono utilizzare più nomi utente e password per ottenere l'accesso

alle risorse per cui hanno i privilegi, è molto probabile che li annotino, indebolendo in questo modo la sicurezza. Ma se hanno una sola password da ricordare, allora difficilmente la appunteranno da qualche parte.

Il single sign-on agevola anche l'amministrazione, garantendo la riduzione del numero di account richiesti per ciascun soggetto.

1.6.2 Svantaggi del single sign-on

Il principale svantaggio dell'utilizzo di SSO è che una volta che un account viene compromesso, l'aggressore ottiene l'accesso illimitato a tutte le risorse autorizzate. Tuttavia, la maggior parte I sistemi SSO includono metodi per proteggere le credenziali dell'utente. Le sezioni seguenti illustrano diversi meccanismi SSO comuni.

1.6.3 Esempi di single sign-on

LDAP e controllo dell'accesso centralizzato

LDAP (Lightweight Directory Access Protocol) è un protocollo software che consente a chiunque di individuare dati relativi a organizzazioni, individui e altre risorse come file e dispositivi in una rete, sia su Internet pubblico che su una intranet aziendale.

Questo protocollo è una versione del Directory Access Protocol (DAP) che fa parte di uno standard per i servizi di directory in una rete, X.500. LDAP è considerata una versione "leggera" (da cui la denominazione "Lightweight") di DAP perché utilizza una minore quantità di codice rispetto ad altri protocolli.

Una directory dice all'utente dove si trova una risorsa nella rete. Sulle reti TCP/IP e, più in generale, su Internet il sistema dei nomi di dominio (DNS) è il sistema di directory comunemente utilizzato per associare il nome di dominio a un indirizzo di rete specifico, ovvero ad una posizione univoca sulla rete. Può anche capitare, però, che l'utente non conosca il nome del dominio. LDAP interviene proprio in situazioni di questo tipo, consentendo a un utente di cercare una risorsa senza sapere dove si trova, anche se per agevolare la ricerca serviranno ulteriori informazioni.

Esiste un'implementazione open source nota come OpenLDAP. LDAP consente di avere un database centralizzato per gli utenti di una organizzazione, che conserva tutte le informazioni rilevanti (i.e. nome utente e password). Si possno aggiungere campi allo schema degli utenti, in modo da averne uno dedicato alla chiave pubblica, che è quella che i server dovrebbero richiedere. LDAP può quindi essere utilizzato in diverse applicazioni o servizi per convalidare gli utenti tramite username e password con server come Docker, Jenkins, OpenVPN e tanti altri.

Gli amministratori di sistema possono usare il single sign-on LDAP per controllare l'accesso a un database LDAP. In questo modo si avrà il vantaggio di sapere chi ha effettuato l'accesso e in che punto. In caso di accessi non autorizzati o violazioni di altro tipo, si potrà risalire, tramite i registri, a chi era connesso nel momento in cui si è verificato l'inconveniente.

Kerberos

Kerberos è è il più comune e conosciuto protocollo di autenticazione di rete che funziona basandosi sui ticket per consentire ai nodi che comunicano su una rete aperta o non sicura di dimostrare reciprocamente la propria identità in modo sicuro. Oggi, in seguito all'introduzione del protocollo in Windows 2000, Kerberos è la tecnologia di autorizzazione standard di Microsoft Windows. Tuttavia, esistono delle implementazioni di Kerberos anche per altri sistemi operativi come Apple OS, FreeBSD, UNIX e Linux. Successivamente è diventato il protocollo standard per i siti web e le implementazioni di single sign-on su varie piattaforme [5].

Kerberos non prevede l'invio di credenziali da un richiedente (client) verso un fornitore (server), ma si basa su chiavi segrete criptate. Il protocollo implica l'introduzione di una terza parte, il Key Distribution Center (KDC) che fornisce ticket basandosi sulla chiave del client e ne verifica la validità per il server. E' proprio questa modalità a tre vie che dà origine al nome del protocollo: il protocollo prende il nome dal personaggio Kerberos (o Cerbero) a partire dalla mitologia greca, il feroce cane da guardia a tre teste di Ade.

In questo modo vengono garantiti gli accessi senza la necessità della solita verifica di username e password, con l'obiettivo primario di dare la possibilità ad utenti non privilegiati di richiedere ticket, memorizzarli e lanciare applicazioni ad essi associate all'interno di un sistema centralizzato.

Vediamo nel dettaglio il fuzionamento del protocollo, facendo riferimento alla Figura 1.5

Il client si autentica in Server di autenticazione (AS) che inoltra il nome utente a un file centro di distribuzione delle chiavi (KDC). Il KDC emette un file biglietto di concessione del biglietto (TGT), che è timestamp e lo crittografa utilizzando il servizio di concessione di biglietti (TGS) chiave segreta e restituisce il risultato crittografato alla workstation dell'utente. Questa operazione viene eseguita raramente, in genere all'accesso dell'utente; il TGT scade a un certo punto sebbene possa essere rinnovato in modo trasparente dal gestore della sessione dell'utente mentre è connesso.

Quando il client ha bisogno di comunicare con un servizio su un altro nodo (un "principal", nel gergo Kerberos), il client invia il TGT al TGS, che di solito condivide lo stesso host del KDC. Il servizio deve essere già stato registrato al

TGS con estensione Nome dell'entità servizio (SPN). Il client utilizza l'SPN per richiedere l'accesso a questo servizio. Dopo aver verificato che il TGT è valido e che l'utente è autorizzato ad accedere al servizio richiesto, il TGS emette il ticket e le chiavi di sessione al client. Il client invia quindi il ticket a server di servizio (SS) insieme alla sua richiesta di servizio.

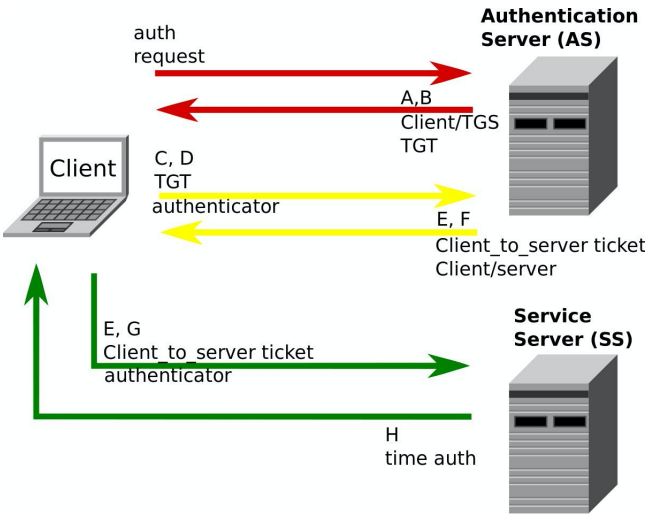


Figura 1.5: Funzionamento del protocollo Kerberos [6].

Gestione dell'identità federata e SSO

Il single sign-on viene utilizzato sulle reti interne ormai da un po' di tempo, ma non su Internet. E' bene sottoloneare però come, a causa del rapido aumento delle applicazioni basate su cloud, siano sempre di più le richieste di una soluzione SSO per gli utenti che accedono alle risorse tramite Internet. La gestione delle identità federata è una forma di SSO viene incontro a questa esigenza.

Uno IAM federato è un sistema per la gestione delle identità e degli accessi a livello di federazione, quindi non più limitata ad una singola organizzazione ma che coinvolge un insieme di organizzazioni. Questo consente a più organizzazioni di aderire ad un'unica federazione e di accordarsi quindi su un metodo per condividere tra loro le identità. Gli utenti di ogni organizzazione, dopo aver effettuato un primo accesso all'interno della propria organizzazione, avranno le credenziali associate all'identità federata. E utilizzando tale identità, potranno accedere alle risorse situate in qualsiasi altra organizzazione che sia all'interno del gruppo.

Una federazione può essere, per esempio, un insieme di reti all'interno di un campus universitario, più campus universitari, più organizzazioni che condivi-

dono le risorse o qualsiasi altro gruppo che necessiti di una gestione comune dell'identità federata.

Una delle sfide che si presentano quando più aziende comunicano all'interno di una stessa federazione è trovare un linguaggio comune. Spesso ciascun sistema opera separatamente dagli altri avendo un funzionamento diverso, ma hanno comunque bisogno di condividere un linguaggio comune. A tal proposito, sistemi IAM federati utilizzano spesso il Security Assertion Markup Language (SAML) e il Service Provisioning Markup Language (SPML).

OAuth e OpenID

OAuth e OpenID sono i due esempi più rilevanti di SSO utilizzati su Internet. OAuth è uno standard aperto progettato per funzionare con HTTP e consente agli utenti di accedere con un accout a più servizi. Ad esempio, gli utenti possono accedere al proprio account Google e utilizzare lo stesso account per accedere a Facebook e a Twitter. Google supporta OAuth 2.0, che non è retrocompatibile con OAuth 1.0. Anche OpenID è uno standard aperto e può essere utilizzato insieme a OAuth o da solo.

1.6.4 In assenza di single sign-on

Può capitare che il SSO non sia disponibile. In questi contesti è importante avere un buon sistema di gestione delle credenziali.

Si tratta di meccanismi che consentono l'amministrazione del ciclo di vita delle credenziali dell'utente (emissione, modifica e revoca) con cui opera un'organizzazione. Tali credenziali fungono da chiavi per svariate piattaforme, strumenti e servizi che i dipendenti di un'organizzazione utilizzano per svolgere i propri ruoli. In altre parole, un sistema di gestione delle credenziali è un custode centralizzato di credenziali, privilegi e politiche per le risorse di un'organizzazione.

Questo sistema fa parte di quella che è nota come infrastruttura a chiave pubblica (PKI): un insieme di ruoli, politiche, hardware, software e procedure per creare, distribuire, utilizzare, archiviare e revocare certificati digitali e gestire la crittografia a chiave pubblica. La PKI è dunque un accordo che lega opportunamente le chiavi pubbliche alle identità di persone e organizzazioni, e che viene delineato dal sistema di gestione delle credenziali per applicare politiche e privilegi di sicurezza.

I sistemi windows, per fare un esempio, sono provvisti di uno strumento di questo tipo. Gli utenti inseriscono le proprie credenziali nel file Credential Manager e, quando necessario, il sistema operativo le recupera e le invia automaticamente. Se gli utenti lo utilizzano per visitare un sito Web e inserire user-

name e password, successivamente, quando rivisiteranno il sito, il Credential Manager riconosce automaticamente l'URL e fornisce le credenziali.

Autenticazione senza single sign-on

Come accennato precedentemente, senza Single Sign-On, ogni sito Web gestisce il proprio database di utenti e le proprie credenziali. Quando si tenta di accedere ad un'app o ad un sito Web, questi ultimi controllano innanzitutto se l'utente è già stato autenticato oppure no. In caso affermativo, viene garantito l'accesso, altrimenti viene chiesto all'utente di accedere.

Figura 1.6: L'utente richiede l'accesso al sito/app [7].

Le credenziali inserite vengono controllate e confrontate con le informazioni presenti nel database utente del sito web o dell'applicazione coinvolta.

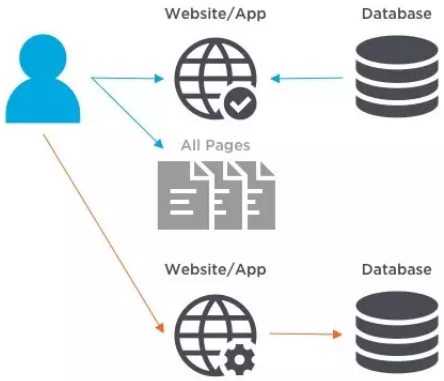


Figura 1.7: L'utente ha avuto l'accesso e richiede l'accesso ad un altro sito [8].

Dopo l'accesso, il sito invia i dati di verifica dell'autenticazione (solitamente sotto forma di cookie o token per tentere traccia della sessione e perchè sono

veloci da elaborare) durante la navigazione nel sito, per verificare di essere autenticati ogni volta che si passa ad una nuova pagina.

Capitolo 2

Caso di studio: Keycloak e SSO

In questo capitolo verrà presentato il contesto aziendale in cui si colloca il progetto, i punti centrali di quest'ultimo e le tecnologie impiegate per la sua realizzazione.

2.1 Scopo della tesi

Scopo di questa tesi è presentare un progetto di backend in collaborazione con Imola Informatica, una società indipendente di consulenza IT. Il lavoro è sostanzialmente diviso in due parti:

- la prima prevede l'installazione e la configurazione di un Identity Manager Keycloak che permetta agli utenti di autenticarsi;
- la seconda parte, invece, riguarda l'impostazione e la configurazione di un SSO (Single Sign-On) tra diverse applicazioni, che permetta, tramite un unico token con protocollo OAuth2, il riconoscimento dell'identità da qualsiasi applicazione senza la ncessità di ulteriori login da parte dell'utente.

2.2 Contesto aziendale

In ambito enterprise normalmente si hanno un'infinità di applicazioni. Imola Informatica è infatti una grande azienda che si serve di più applicazioni e servizi. Questo porta i dipendenti a dover accedere a più app con set di credenziali diverse. Il SSO serve a prevenire questo problema, centralizzando di fatto la gestione degli utenti e diminuendo drasticamente la proliferazione di account in rete. In quest'ottica, si è palesata la necessità di creare un sistema di Single Sign On in modo da limitare l'uso delle credenziali dell'utente ad un

perimetro più circoscritto. Sono state quindi poste le basi per la realizzazione di un sistema che seguisse il principio del least privilege e che, allo stesso tempo, garantisse la limitazione del perimetro di attacco.

2.2.1 Least privilege

L'idea di offrire ai dipendenti l'accesso più "basso" possibile per svolgere il proprio lavoro è chiamata principio del privilegio minimo, Principle of Least Privilege, in inglese (PoLP). E' importante tanto in ambito aziendale, quanto nella vita quotidiana. Il PoLP è un componente essenziale per la gestione degli accessi privilegiati (PAM).

Il least privilege offre agli utenti il minimo livello di accesso alle risorse aziendali quali server, applicazioni e dispositivi, senza però compromettere il loro lavoro e la produttività.

Ogni dipendente, indipendentemente dal tipo di account, dovrebbe disporre dell'accesso minimo necessario alle risorse aziendali. Ciò che varia sarà la modalità di accesso, in base alla tipologia di account.

Gli account privilegiati (chiamati anche account super-utente) hanno un accesso che va al di là delle opzioni utente standard. Basti pensare all'utente amministratore (admin), che ha la possibilità di accedere a informazioni aziendali riservate o di inviare aggiornamenti da remoto a più dispositivi utente. Questi account possono appartenere alla dirigenza esecutiva o ai team IT.

Gli account non privilegiati (detti account standard), invece, hanno solo un accesso di base ai server e alle applicazioni necessarie per svolgere il proprio lavoro.

Sebbene gli account privilegiati e non privilegiati siano diversi, i principi del least privilege devono essere applicati a tutti gli account utente dell'organizzazione, senza distinzioni.

I vantaggi del least privilege possono essere riassunti come riportato di seguito:

- *riduce la superficie esposta agli attacchi informatici.* Oggigiorno, gli attacchi più avanzati sono basati sullo sfruttamento delle credenziali privilegiate. Limitando i privilegi che garantiscono accesso illimitato ai sistemi target (super-user o amministratore), l'utilizzo del least privilege contribuisce a ridurre la porzione totale esposta agli attacchi informatici;

- *arresta la diffusione di malware.* Applicando il principio del privilegio minimo agli endpoint, gli attacchi malware (i.e. SQL injection) non possono sfruttare i privilegi elevati per ottenere un livello di accesso più alto, al fine di installare o eseguire malware o di danneggiare addirittura la macchina;

- *migliora la produttività degli utenti finali.* Revocare i diritti di amministratori locali degli utenti aziendali contribuisce a ridurre i rischi, mentre consentire di elevare a livelli superiori i privilegi just-in-time in base alle esigenze preserva la produttività degli utenti, e minimizza le chiamate all'helpdesk IT;

- *contribuisce a razionalizzare la conformità e le verifiche.* Molte policy interne e diversi requisiti normativi obbligano le organizzazioni a implementare il principio del least privilge sugli account privilegiati per prevenire danni ai sistemi critici, intenzionali o meno. L'applicazione del privilegio minimo aiuta le organizzazioni a dimostrare la conformità, grazie al percorso completo di audit delle attività privilegiate.

2.2.2 Perimetro di attacco ridotto

Partendo dal presupposto che il least privilege garantisce che il minor numero possibile di persone abbia accesso alle applicazioni e ai dati più sensibili di un'azienda, è chiaro che meno persone possono accedervi, minori sono le opportunità per i criminali informatici di compromettere un'identità.

Questo fatto è diventato particolarmente importante nella maggior parte degli ambienti aziendali remoti odierni. Secondo il Data Breach Investigations Report 2021 di Verizon [9], il 61% di tutte le violazioni dei dati nel 2021 è avvenuto a causa di credenziali compromesse. Attacchi come questi stanno diventando sempre più comuni e sebbene l'implementazione del least privilege di per sé non garantisca che un'organizzazione sia al sicuro dagli attacchi informatici, riduce significativamente il danno che un hacker può infliggere.

2.3 Keycloak

Keycloak è uno strumento open source sfruttato dai sistemi IAM, in partcolare è impiegato in applicazioni moderne come applicazioni single-page, applicazioni mobile ed API REST. Si tratta di un progetto la cui prima versione è stata realizzata nel settembre del 2014, con l'obiettivo di rendere più facile per gli sviluppatori proteggere le proprie applicazioni. Nel 2016 Red Hat ha cambiato il prodotto RH SSO dall'essere basato sul framework PicketLink ad essere basato sul Keycloak upstream Project. Ciò ha seguito l'unione del codice base PicketLink in Keycloak. Da allora è diventato un progetto open source consolidato con una forte e sempre più numerosa community di utenti.

Viene utilizzato in innumerevoli contesti, vale a dire per scenari che vanno da piccoli siti Web con solo una minima selezione di utenti fino a grandi imprese con milioni di utenti.

Keycloak fornisce pagine di login completamente personalizzabili, inclusa l'autenticazione a due o più fattori, procedure per il recupero delle password, notificazione agli utenti per aggiornare regolarmente le password o per accettare termini e condizioni e molto altro ancora. Tutto questo viene offerto senza che ci sia bisogno di apportare modifiche alla propria applicazione.

Delegando l'autenticazione a Keycloak, le applicazioni non devono preoccuparsi dei diversi meccanismi di autenticazione o di come archiviare in modo sicuro le password.

Questo approccio infatti garantisce un più alto livello di sicurezza in quanto le applicazioni non hanno accesso diretto alle credenziali degli utenti, essendo invece dotate di token di sicurezza che danno loro accesso solo a ciò di cui hanno bisogno.

Keycloak fornisce il single sign-on e le funzionalità per la gestione delle sessioni, consentendo agli utenti di accedere a più applicazioni, autenticandosi una sola volta. Sia gli utenti che gli amministratori possono vedere in qualsiasi momento dove sono autenticati gli utenti e possono terminare da remoto le sessioni quando richiesto.

Il motivo per cui Keycloak è così gettonato è perchè si tratta una soluzione leggera, facile da installare ed è altamente scalabile. Supporta inoltre il clustering in più data center garantendo un'ampia disponibilità di spazio e risorse.

2.3.1 Funzionalità di Keycloak

Vediamo ora nel dettaglio le principali funzionalità di Keycloak, in modo da comprenderne meglio la sua importanza.

- *Supporto di più protocolli*: keycloak supporta i 3 protocolli standard del settore, ossia OpenID Connect, OAuth 2.0 e SAML 2.0. Questo è importante dal punto di vista della sicurezza e semplifica inoltre l'integrazione con le applicazioni esistenti e nuove.

- *SSO*: come accennato precedentemente, Keycloak supporta il single sign-on e anche il single sign-out.

- *Console Admin*: Keycloak mette a disposizione una GUI web con cui interagire per selezionare tutte le configurazioni richieste per svolgere l'attività desiderata.

- *Identità e accesso dell'utente*: keycloak è uno strumento per la gestione delle identità e degli accessi, quindi consente di creare un database utente con ruoli e gruppi personalizzati. Sarà possibile sfruttare ulteriormente

queste informazioni per autenticare gli utenti all'interno dell'applicazione e proteggere alcune parti e determinate risorse in base ai ruoli predefiniti.

- *Intermediario di identità*: keycloak può assumere il ruolo di proxy tra gli utenti di una organizzazione e alcuni provider di identità esterni.

  Grazie a questa sua capacità di intermediazione è possibile collegare interi database di utenti esistenti dai social network o da altri provider di identità aziendali. Può persino integrarsi con delle directory utente esistenti, come Active Directory e i server LDAP.

- *Social Identity*: la caratteristica più importante di Keycloak è che consente di utilizzare i Social Identity provider. Ha infatti il supporto integrato per Facebook, Google, Twitter, Stack Overflow e gitHub. Occorre comunque configurarli tutti manualmente dal pannello di amministrazione. Nella documentazione di Keycloak, si trova l'elenco completo dei provider di identità social supportati e il relativo manuale di configurazione.

- *Personalizzazione delle pagine* : è possibile personalizzare tutte le pagine visualizzate dagli utenti. Essendo queste in formato .ftl, si possono utilizzare i classici markup HTML e i fogli di stile CSS per adattare la pagina allo stile della propria applicazione e al marchio dell'organizzazione. In più sono supportati anche degli script JS per la customizzazione.

2.4 Single sign-on

2.4.1 Come funziona il single sign-on

L'autenticazione con SSO si basa su una relazione di fiducia tra domini (siti Web). Con il Single Sign-On, quando si tenta di accedere ad un'applicazione oppure ad un sito Web, quest'ultimo verifica innanzitutto se l'utente è già stato autenticato oppure no. In caso affermativo, viene consentito l'accesso al sito, altrimenti viene inviata una soluzione SSO per l'accesso.

Si inseriscono le credenziali aziendali e il sistema richiede l'autenticazione dal provider di identità o dal sistema di autenticazione utilizzato dall'azienda. Dopo aver verificato l'identità viene notificata la soluzione SSO che passa i dati di autenticazione al sito Web e consente di tornare a tale sito.

Dopo l'accesso, il sito invia i dati di verifica dell'autenticazione con l'utente mentre ci si sposta all'esterno del sito per verificare di essere autenticati ogni volta che si passa a una nuova pagina.

Con il single sign-on i dati di verifica dell'autenticazione assumono la forma di token. Nelle due figure seguenti verranno esplicitati questi passaggi.

Figura 2.1: L'utente richiede l'accesso [10].

Il sito web reindirizza l'utente alla soluzione SSO per accedere che verifica il provider di identità dell'utente.

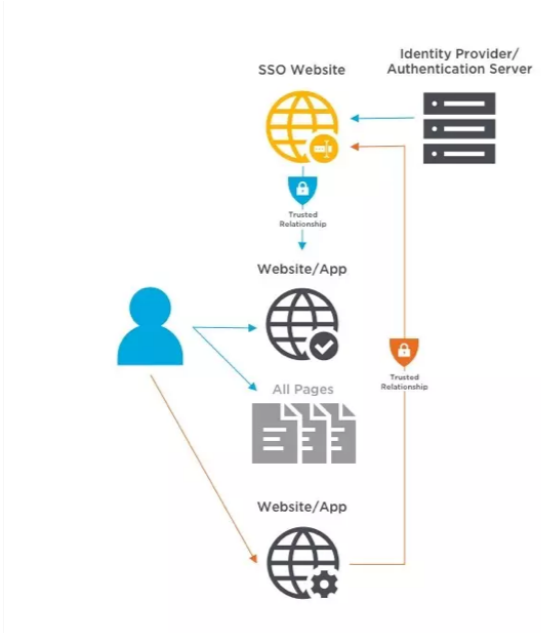


Figura 2.2: L'utente ottiene l'accesso e fa richiesta per un altro sito [11].

Quando l'utente tenta di accedere ad un sito Web diverso, questo verifica l'identità dell'utente con la soluzione SSO senza richiedere un accesso aggiuntivo,

dato che l'utente è stato già autenticato.

2.5 Social Login

L'accesso social, più comunemente noto come social login, è una forma di Single Sign-On che prevede l'utilizzo di informazioni esistenti da un file servizio di social networking (ad esempio Facebook, Twitter o Google) per accedere a un sito Web di terze parti invece di creare un nuovo account di accesso specifico per quel sito Web.

È progettato per semplificare gli accessi per gli utenti finali e per fornire informazioni demografiche sempre più affidabili agli sviluppatori web.

2.5.1 Come funziona il social login

Il social login collega gli account di uno o più servizi di social network a un sito web, in genere utilizzando un plug-in o un widget.

Selezionando il servizio di social networking desiderato, l'utente utilizza semplicemente le proprie credenziali per quel servizio per accedere al sito web. Questo, a sua volta, evita all'utente finale di ricordare le informazioni di accesso per più e-commerce e siti Web fornendo ai proprietari del sito informazioni demografiche uniformi tramite il servizio di social networking.

Molti siti che supportano il social login offrono anche una registrazione online tradizionale per coloro che lo desiderano o che non hanno un account con un servizio di social networking compatibile e a cui quindi sarebbe preclusa la creazione di un account con il sito web.

2.5.2 Vantaggi del social login

Gli studi hanno dimostrato che i moduli di registrazione del sito web sono inefficienti poiché molte persone forniscono dati falsi, dimenticano le informazioni di accesso al sito o semplicemente rifiutano di registrarsi in primo luogo. Uno studio condotto nel 2011 da Janrain e Blue Research ha rilevato che il 77% dei consumatori preferisce l'accesso social come mezzo di autenticazione rispetto ai metodi di registrazione online più tradizionali [12].

Benefici addizionali:

- **contenuti mirati**: i siti web possono ottenere un profilo e i dati del grafico sociale al fine di indirizzare contenuti personalizzati all'utente. Ciò include informazioni come nome, e-mail, città natale, interessi, attività e amici. Tuttavia, ciò può creare problemi per la privacy e ridurre la varietà di visualizzazioni e opzioni disponibili su Internet;

- **identità multiple**: gli utenti possono accedere a siti web con più identità sociali che consentono loro di controllare meglio la propria identità online;

- **dati di registrazione**: mlti siti web utilizzano i dati del profilo restituiti dall'accesso ai social invece di consentire agli utenti di inserire manualmente i propri PII (Informazioni di identificazione personale) nei moduli web. Questo può potenzialmente accelerare la registrazione o il processo di iscrizione;

- **e-mail preconvalidata**: un provider di identità che supportano la posta elettronica come Google e Yahoo! può restituire l'indirizzo e-mail dell'utente al sito web di terze parti impedendo all'utente di fornire un indirizzo e-mail inventato durante il processo di registrazione;

- **Collegamento dell'account**: poiché l'accesso social può essere utilizzato per l'autenticazione, molti siti Web consentono agli utenti precedenti di collegare l'account del sito preesistente al proprio account di accesso social senza forzare la nuova registrazione.

2.5.3 Svantaggi del social login

L'utilizzo dell'accesso social tramite piattaforme come Facebook potrebbe rendere involontariamente inutili i siti Web di terzi all'interno di determinate biblioteche, scuole o luoghi di lavoro che bloccano servizi di social networking per motivi di produttività.

Può anche causare difficoltà nei paesi con active censura regimi, come Cina e la sua "Progetto Scudo d'Oro" in cui il sito web di terze parti potrebbe non essere censurato attivamente, ma viene effettivamente bloccato se l'accesso ai social di un utente viene bloccato.

Ci sono molti altri rischi che derivano dall'utilizzo degli strumenti di accesso social. Questi accessi sono anche una nuova frontiera per le frodi e l'abuso di account poiché gli aggressori utilizzano mezzi sofisticati per hackerare questi meccanismi di autenticazione. Ciò può comportare un aumento indesiderato delle creazioni fraudolente di account o peggio, malintenzionati che rubano con successo le credenziali degli account dei social media da utenti legittimi.

Un modo in cui gli account dei social media vengono sfruttati è quando gli utenti sono invogliati a scaricare estensioni del browser dannose che richiedono autorizzazioni di lettura e scrittura su tutti i siti web. Questi utenti non sono consapevoli del fatto che in seguito, in genere circa una settimana dopo l'installazione, le estensioni scaricheranno del malware Javascript in background dal suo sito di comando e controllo da eseguire sul browser dell'utente. Da quel

momento in poi, questi browser infetti da malware possono essere efficacemente controllati da remoto. Queste estensioni attenderanno quindi fino a quando l'utente accederà a un social media o a un altro account online e utilizzando tali token o credenziali si registrerà per altri account online senza l'espressa autorizzazione dell'utente legittimo.

2.6 Social login e protocollo OAuth2

Il diagramma di sequenza illusrato in Figura 2.3 spiega cosa avviene a partire dal momento in cui un utente richede l'accesso ad un sito web mediante social login, fino a quando non gli viene fornito.

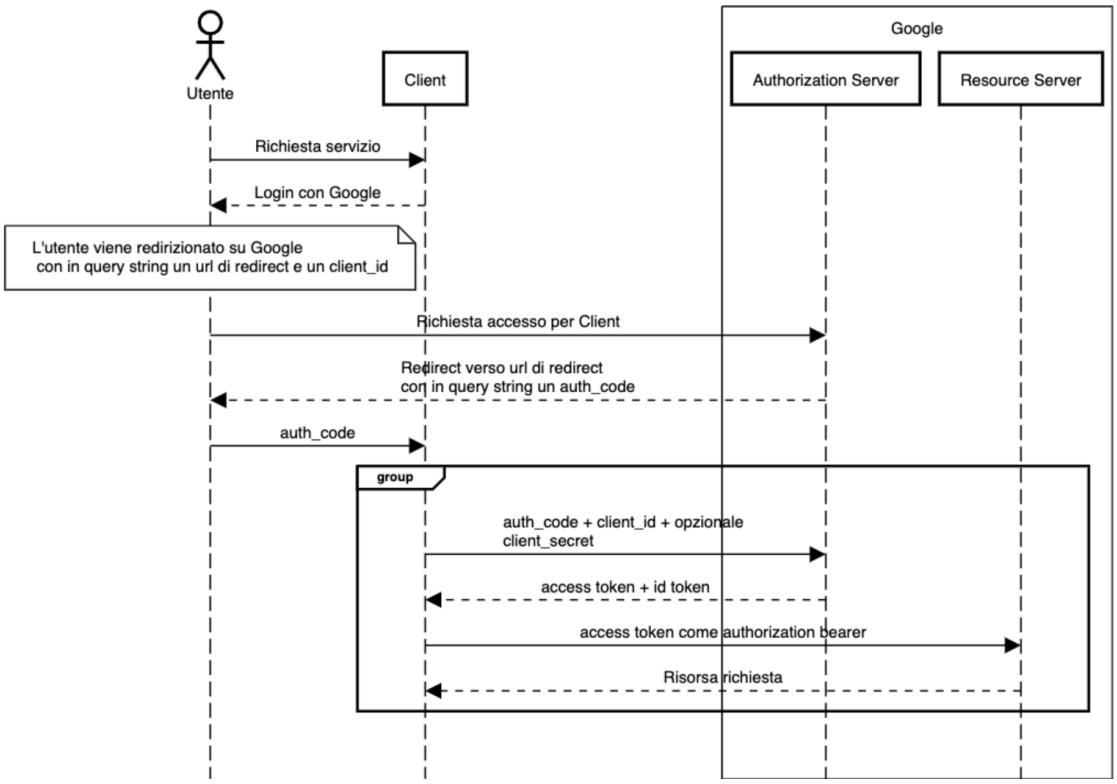


Figura 2.3: Social Login con protocollo OAuth2.

OAuth 2.0 è un protocollo standard aperto che consente alle applicazioni di accedere alle risorse protette di un servizio da parte dell'utente [13]. Viene ampiamente utilizzato in applicazioni native, applicazioni web e dispositivi mobili.

Sono molte le aziende che offrono endpoint OAuth; tra le più note ed importanti vanno menzionate Google, Facebook, Linkedin, Github, ma il numero è in crescita.

Diamo ora una rappresentazione ad alto livello del protocollo. Il processo di autorizzazione si descrive come segue:

1. l'applicazione invia una richiesta di autorizzazione per accedere ad una risorsa protetta ad un Authorization Server;

2. il proprietario della risorsa concede l'accesso;

3. l'Authorization Server restituisce un Access Token da utilizzare in tutte le successive richieste come una sorta di "targhetta di riconoscimento".

In OAuth 2.0, il termine "Grant type", "tipo di concessione", si riferisce al modo in cui un'applicazione ottiene un Access Token (token di accesso).

Il suddetto protocollo definisce definisce diversi Grant type, ognuno dei quali è ottimizzato per un particolare caso d'uso, che si tratti di un'app Web, un'app nativa, un dispositivo senza la possibilità di avviare un browser Web o applicazioni server-server.

La scelta del grant type adatto al proprio caso non dipende solo dal tipo di applicazione, ma anche da altri parametri come il livello di fiducia per il cliente o l'esperienza che si vuole garantire ai propri utenti.

I più noti ediffusi sono il Client credentials grant e l'Authorization code grant, che è quello utilizzato all'interno del progetto. Le estensioni di OAuth 2.0 possono definirne degli altri.

Client Credentials

Il Client Credentials grant viene utilizzato principalmente in applicazioni intra-server, come servizi batch o applicazioni a riga di comando.

In questo caso l'Authorization Server concede l'accesso all'applicazione, invece che all'utente. Per utilizzare questo flusso, all'applicazione deve essere stato assegnato un client ID e un client secret. Questi parametri sono generati dall'Authorization Server e sono necessari per garantire che il client che si connette sia effettivamente chi dice di essere.

Vediamo prima uno schema del flusso e diamone in seguito una descrizione esaustiva.

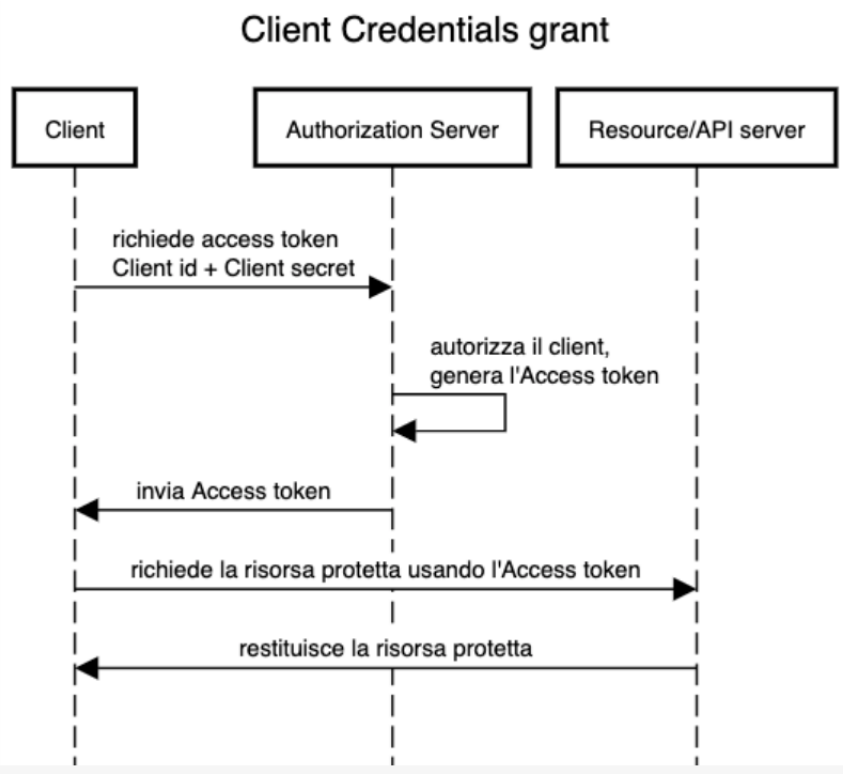


Figura 2.4: Diagramma di sequenza del processo del Client Credential.

- il client (i.e. l'applicazione) invia una richiesta di autorizzazione insieme ai parametri `client_id` e `client_secret` all'Authorization Server;
- l'Authorization Server verifica le credenziali fornite e restituisce al client un Access Token ed eventualmente un Referesh Token;
- l'applicazione/client utilizza l'Access Token in ogni successiva richiesta al servizio.

Authorization Code

L'Authorization Code è il flusso più usato, in quanto viene impiegato in applicazioni mobile e web che operano effettivamente su risorse che appartengono all'utente.

Si differenzia dalla maggior parte degli altri grant type richiedendo prima all'app di avviare un browser per dare inizio al flusso.

Per poterlo utilizzare, l'applicazione deve essere stata preventivamente registrata presso l'Authorization Server, fornendo almeno un nome e un redirect URI, che verrà utilizzato per informare l'applicazione che l'utente ha autorizzato la richiesta.

Durante la registrazione, il provider assegna all'applicazione un `client_id` e un `client_secret`, che verranno scambiati durante il processo di autorizzazione.

Per accedere ad una determinata risorsa, l'applicazione deve indicare l'elenco delle "autorizzazioni" richieste, dette `scopes`. Gli scopes sono definiti dal Resource Server e sono normalmente inclusi nella documentazione del servizio.

Diamo ora una descrizione dettagliata del funzionamento del processo illustrato in Figura 2.5.

- l'utente apre l'applicazione;
- l'applicazione apre un'istanza del browser e la indirizza tramite URI all'endpoint di autorizzazione presso l'Authorization Server, passandogli sia il client ID che l'elenco degli `scopes`;
- poi l'utente si autentica e concede l'accesso alle risorse (autorizzazione);
- l'Authorization Server, dopo aver verificato le credenziali dell'utente, reindirizza il client (applicazione) al redirect URI allegando un Authorization Token;
- l'applicazione effettua una richiesta di generazione dell'Access Token, passando l'Authorization Token e il `client_secret`;
- Il server convalida il `client_secret` e l'Authorization Token, genera e restituisce un Access Token. Eventualmente allega anche un Refresh Token;
- a questo punto l'applicazione può utilizzare l'Access Token per ogni successiva richiesta al servizio in modo che non debba più chiedere la verifica delle credenziali.

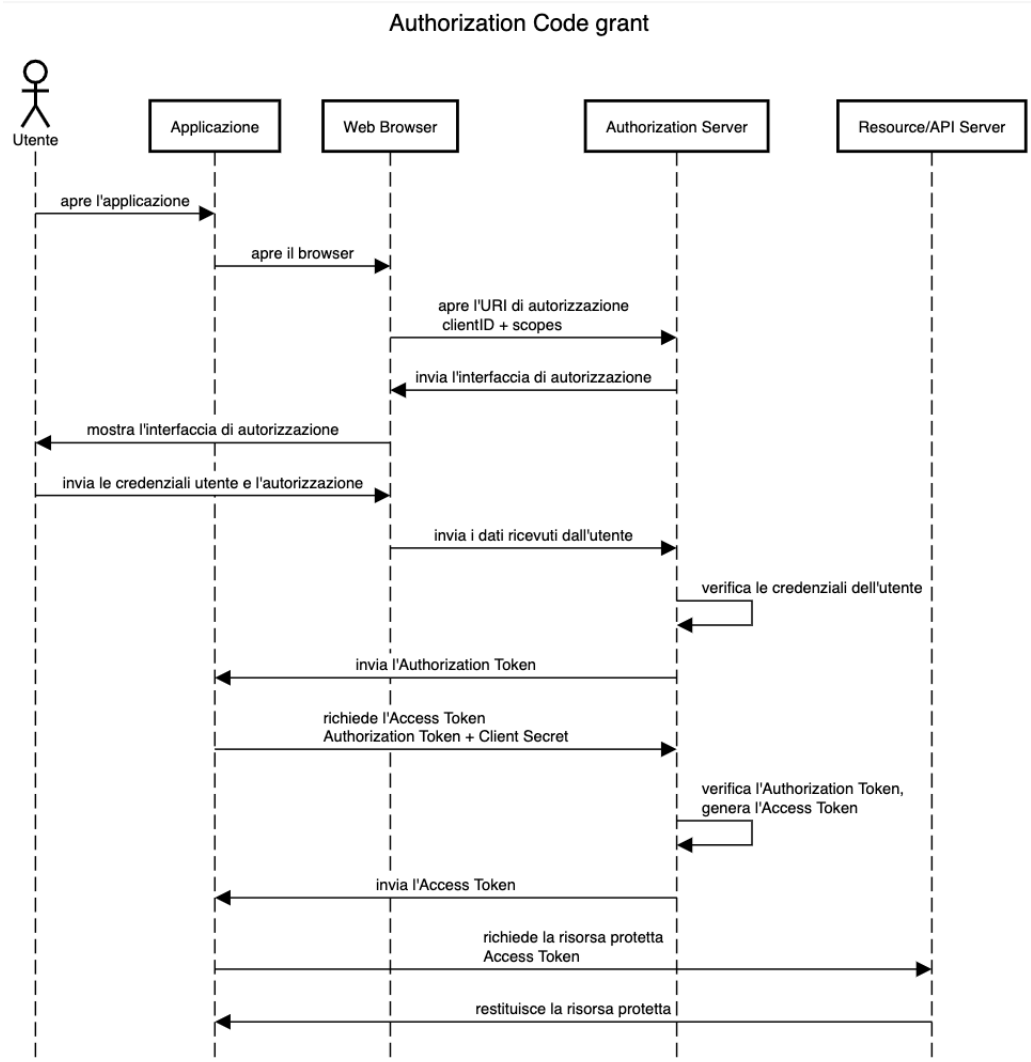


Figura 2.5: Diagramma di sequenza per il funzionamento dell'Authorization Code grant.

2.7 JWT - Json Web Token

Un servizio REST dovrebbe essere, per definizione, stateless. Vale a dire che le interazioni tra client e server devono essere senza stato, e rispettare così la comunicazione nativa del protocollo HTTP che è, appunto, stateless. L'eventuale gestione dello stato avviene sul client, in modo da ottimizzare le prestazioni del server che non dovrà curarsi dello stato. L'approccio stateless di un'architettura RESTful rende inadatto l'utilizzo dei cookie di sessione per il meccanismo di autenticazione.

E' proprio qui che entra in gioco il JSON Web Token (JWT), un nuovo standard aperto definito nella RFC 7519 [14]. Adottato sempre più di frequente da svariati siti web e applicazioni, il JWT è una delle tecnologie che sono state impiegate contestualmente al progetto discusso in questa sede.

Un JSON Web Token è un token di accesso che consente lo scambio sicuro di dati tra due parti. Contenendo tutte le informazioni rilevanti su un'entità, non necessita di alcuna interrogazione al database e non richiede nemmeno che la sessione debba essere memorizzata sul server (stateless session, appunto).

I JWT sono quindi largamente utilizzati nei processi di autenticazione. I brevi messaggi possono essere criptati e quindi fornire informazioni sicure su chi è il mittente e se questo sia dotato dei permessi di accesso necessari. Gli utenti entrano in contatto con il token solo indirettamente, ad esempio inserendo il proprio nome utente e la password in una form. La comunicazione effettiva avviene tra le varie applicazioni lato client e server.

2.7.1 Quando usare JWT

Di seguito vengono descritti due scenari che prevedono l'utilizzo di JWT.

- *Autorizzazione*: è lo scenario più comune per l'utilizzo di JWT. Non appena l'utente effettua l'accesso, ogni richiesta successiva includerà il JWT, e questo consentirà all'utente di accedere a percorsi, servizi e risorse consentiti per quello specifico token;

- *Scambio di informazioni*: i JSON Web token sono un ottimo modo per trasmettere in maniera sicuro le informazioni tra le parti. Grazie alla capacità dei JWT di poter essere firmati, ad esempio utilizzando coppie di chiavi pubblica/privata, è fortemente assicurato che i mittenti siano chi dichiarano di essere. Inoltre, poiché la firma viene calcolata utilizzando l'intestazione e il payload, si può addirittura verificare che il contenuto non sia stato manomesso. Di questo se ne discuterà in dettaglio nei prossimi paragrafi.

2.7.2 Perchè usare JWT

La documentazione di JWT fornisce un ottimo spunto per comprendere i motivi che hanno portato alla scelta di adottare questo tipo di token nella maggior parte delle applicazioni [15]. A tal proposito, avendo a disposizione altre tipologie di token come i Simple Web Tokens (SWT) o i Security Assertion Markup Language Tokens (SAML), vediamo ora i vantaggi dell'utilizzo dei JSON Web Tokens (JWT).

Poiché JSON è meno verboso di XML, anche quando è codificato le sue dimensioni sono inferiori, rendendo JWT più compatto di SAML. E questo fa di JWT una buona scelta da adottare soprattutto negli ambienti HTML e HTTP.

Dal punto di vista della sicurezza, SWT può essere firmato simmetricamente solo da un secret condiviso tramite l'algoritmo HMAC. I token JWT e SAML possono possono invece utilizzare una coppia di chiavi pubblica/privata sotto forma di certificato X.509 (definito nella RFC5280 [16]) per la firma.

Firmare un file XML con firma digitale XML senza introdurre grosse falle di sicurezza è molto difficile e spesso si predilige il formato JSON anche per la semplicità della firma rispetto a XML.

In aggiunta, i parser JSON sono comuni nella maggior parte dei linguaggi di programmazione perché mappano direttamente gli oggetti. Al contrario, XML non ha una mappatura naturale da documento a oggetto e risulta quindi più facile lavorare con JWT anziché con SAML.

Infine, per quanto riguarda l'utilizzo, JWT è largamente diffuso su Internet. Questo sottolinea la facilità di elaborazione lato client del token su più piattaforme, con particolare riferimento ai dispositivi mobili.

2.7.3 Struttura di un JWT

Nella sua forma compatta, un JSON Web token è costituito da tre parti separate da un punto (`.`) :

- *Header*;
- *Payload*;
- *Signature*.

Un JWT assume quindi una forma del tipo:

`xxxxx.yyyyy.zzzzz`

Header

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

Listato 2.1: header del JWT

L'`header` in genere consiste di due parti: il tipo di token, JWT in questo caso, e l'algoritmo di firma utilizzato. Tipicamente si usano i seguenti algoritmi:

- **RS256**: *algoritmo asimmetrico* basato su una chiave privata per la generazione della firma e una chiave pubblica per la sua validazione;
- **HS256**: *algoritmo simmetrico* in cui la stessa chiave, condivisa in modo sicuro tra le parti, è usata per generare e validare la firma.

Payload

```
{
    "sub": "1234567890",
    "name": "Sara Ragnetti",
    "admin": true
},
```

Listato 2.2: payload del JWT

La seconda parte del token è il payload, che contiene i cosiddetti "claims". Si tratta di dichiarazioni su un'entità (i.e. un utente) e alcune informazioni aggiuntive. Esistono tre tipi di claims: registrati, pubblici e privati.

- *registered claims*, sono un insieme di claims predefiniti registrati all'interno dello IANA JSON Web Token Claim Register [17]. Il loro utilizzo è fortemente consigliato, ma non obbligatorio. Alcuni esempi sono l'emittente del token ("iss" per issuer), il dominio di destinazione ("aud" per audience) e il tempo di scadenza ("exp" per expiration time). Il motivo per cui sono state utilizzate delle abbreviazioni per i nomi dei claims è per mantenere la lunghezza del token il più breve possibile;
- *public claims*, i claim pubblici possono essere definiti a piacimento da chiunque utilizzi i JWT. Tuttavia, per evitare collisioni nella semantica delle keys, dovrebbero essere registrati pubblicamente nello IANA JSON Web Token Claim Register o essere definiti come URI contenenti uno spazio dei nomi resistente alle collisioni;
- *private claims*, sono dei claims privati e "personalizzati" che vengono creati per condividere informazioni tra le parti che concordano di utilizzarle. Mentre i claims pubblici contengono informazioni come "name" o "e-mail", quelli privati sono più specifici. Un'informazione tipica è l'ID utente. Non sono nè registrate, nè pubbliche, ma è importante assicurarsi che il nome non sia in conflitto con gli altri tipi di claim.

In generale, i payload possono contenere un numero arbitrario di claim, ma è consigliato limitare le informazioni nel token allo stretto necessario. Più grande è il JWT, e più risorse richiede per la decodifica.

Signature

La terza ultima parte è la signature, ovvero la firma. Per crearla è necessario avere l'header e il payload codificati in Base64, un secret, e metodo di firma/codifica specificato nell'intestazione.

Ad esempio se si desidera utilizzare l'algoritmo HMAC SHA256, la firma verrà creata come riportato di seguito:

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

Listato 2.3: signature del JWT

La struttura è definita dalla JSON Web Signature (JWS), uno standard secondo RFC 7515 [18]. Affinché la firma funzioni, è necessario utilizzare una chiave segreta nota solo all'applicazione originale. Da un lato, questa firma verifica che il messaggio non sia stato modificato lungo il percorso e dall'altro, se il token è firmato con una chiave privata, assicura che il mittente del JWT sia chi dichiara di essere.

A seconda della sensibilità e quindi dell'importanza dei dati, ci sono diverse procedure per l'apposizione della signature:

- *nessun backup*, se il fattore di protezione dei dati è basso, può essere specificato il valore "none" nell'header. In questo caso non viene generata alcuna firma e il JSON Web Token consiste solo di 2 parti: header e payload. Non avendo alcuna protezione, il payload è leggibile in chiaro in seguito alla decodifica e non viene nemmeno verificato se il messaggio provenga dal mittente corretto o se sia stato modificato durante il tragitto;

- *firma (JWS)* solitamente è sufficiente controllare se i dati provengano dal giusto mittente e se siano stati modificati lungo il percorso. Per fare ciò si utilizza lo schema JSON Web Signature (JWS), che assicura che il messaggio non sia stato modificato e che provenga dal mittente corretto. Con questa procedura, il payload può essere letto in chiaro anche dopo la decrittazione;

- *firma (JWS) e crittografia (JWE)*, è possibile utilizzare una JSON Web Encryption (JWE) in aggiunta al JWS. JWE cripta il contenuto del payload, che viene successivamente firmato con JWS. Per decifrare il contenuto, viene specificata una password comune o una chiave privata. Il mittente viene quindi verificato, il messaggio è riservato e autentico e il payload non può essere letto in chiaro dopo la decrittazione.

Nella figura seguente è raffigurato, da un lato, un JSON Web token nella forma di codice JSON e, dall'altro, la forma codificata in base64 delle tre parti che costituiscono tale token concatenate da un punto.

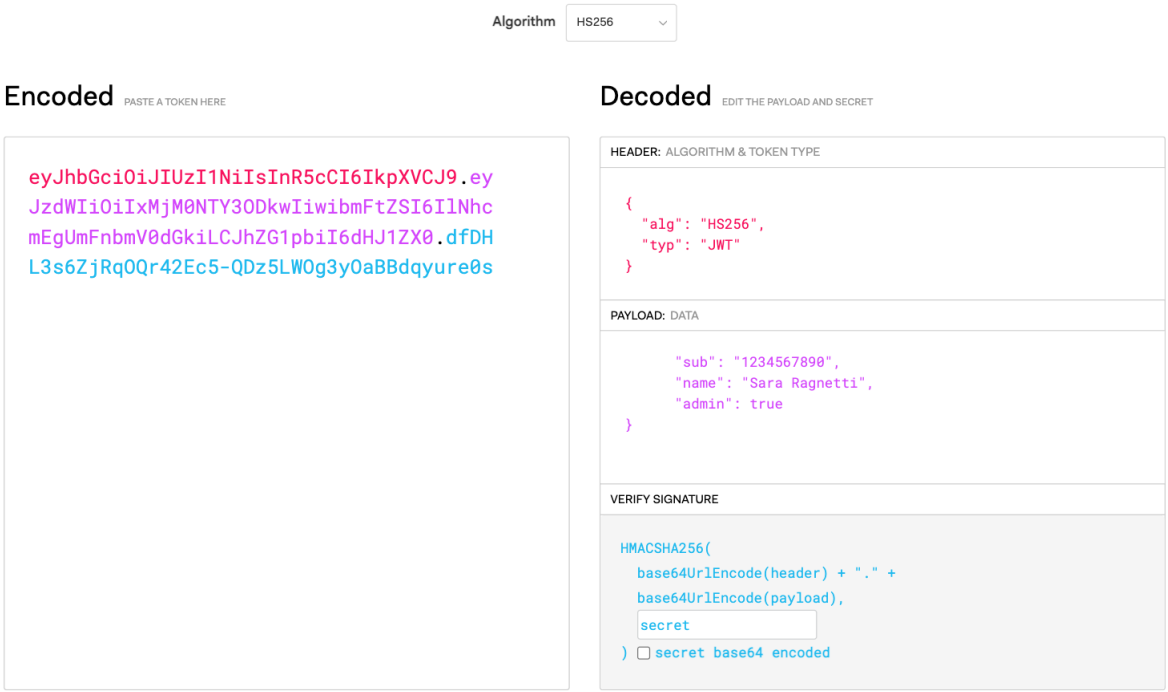


Figura 2.6: Schermata di jwt.io con la decodifica di un token affiancata al codice JSON da cui è ricavata.

Capitolo 3

Il progetto

3.1 Strumenti utilizzati

3.1.1 SpringBoot

Spring è un progetto open source che fornisce un approccio modulare semplificato per la realizzazione di app con Java. I progetti Spring si sono diffusi a partire dal 2003 in risposta alle complessità dello sviluppo Java iniziale per fornire supporto allo sviluppo di app Java. Il nome Spring fa riferimento al framework dell'applicazione o all'intero gruppo di progetti e moduli. Spring Boot è invece un modulo specifico, creato come estensione di Spring.

Si tratta di una soluzione "convention over configuration" per il framework Spring di Java, vale a dire una filosofia e una tecnica di progettazione orientate allo sviluppo di programmi secondo convenzioni e impostazioni predefinite, che limitano il numero di decisioni da parte del programmatore, ma garantendo comunque una certa flessibilità dando la possibilità di sovrascrivere le scelte di default mediante delle configurazioni esplicite. In altre parole, Spring Boot è uno strumento che semplifica e accelera lo sviluppo di app Web e microservizi Java ed è il motivo per cui la maggior parte delle nuove applicazioni Spring sono basate anche su Spring Boot.

La flessibilità di SpringBoot ha fatto sì che venisse individuato come framework principale all'interno del contesto aziendale per la realizzazione del progetto di tesi.

Caratteristiche di Spring Boot

Sebbene le caratteristiche di cui sopra siano ciò che contraddistingue Spring Boot da altri framework, le sue peculiarità non finiscono qui.

Si può infatti prediligere Spring Boot a causa della sua flessibilità nella scelta di configurazioni XML, Java Beans e Transaction Database. Fornisce anche una forte elaborazione batch e gestisce persino gli endpoint REST in tempi ridotti. Può, inoltre, facilitare la gestione delle dipendenze e viene fornito con Embedded Servlet Container.

Vantaggi di Spring Boot

Spring Boot offre diversi vantaggi.

- *Spring Boot funziona bene con diversi servlet containers.* Utilizza Tomcat per impostazione predefinita, ma si può facilmente sostituire con Jetty, Undertow, Resin e Wildfly, scegliendo in base alle specifiche funzionalità interessate. In più, Spring Boot identifica automaticamente il servlet impostato come nuovo predefinito durante la fase di avvio. Questi fattori offrono la flessibilità di scegliere i server integrati più adatti alle proprie esigenze;

- *riduzione del codice boilerplate*, il database in memoria e il server incorporato (Tomcat) di Spring Boot riducono o eliminano il codice boilerplate tipicamente necessario per configurare un'applicazione. In questo modo, i programmatori possono abbreviare i tempi di sviluppo e i cicli di aggiornamento, rendendo gli utenti più soddisfatti e, allo stesso modo, i dipendenti più produttivi. È uno dei vantaggi di Spring Boot che aiuta gli sviluppatori a risparmiare tempo;

- *nessuna configurazione XML richiesta.* Gli sviluppatori di progetti Spring possono scegliere di utilizzare annotazioni o configurazioni XML, ma anche di non farlo, evitando così di eseguire i passaggi aggiuntivi richiesti;

- *i file WAR non sono richiesti.* Nonostante Spring Boot possa utilizzare i file WAR (Web Application Resource), questi non sono necessari. Tuttavia, può fare affidamento sui JAR (Java resource). Tali file hanno una struttura più breve e più semplice che li rende utili sia per gli sviluppatori che per gli utenti. I file leggeri funzionano rapidamente per connettere le applicazioni con gli strumenti di cui necessitano per funzionare. L'opzione di utilizzare WAR o JAR avvantaggia anche i team di sviluppo. Se qualcuno nel gruppo non ha esperienza con JAR, può fare affidamento su WAR, e viceversa;

- *gestione delle dipendenze tramite POM.* Spring Boot non obbliga a utilizzare un POM padre (modello a oggetti del progetto). L'aggiunta dell'artefatto `Spring-boot-dependencies`, infatti, consente di gestire le dipendenze senza fare affidamento su un file POM o XML padre;

- *una grande community.* Come la maggior parte degli strumenti open source, Spring Boot ha una vasta comunità di utenti formata da persone che amano condividere le proprie intuizioni e creazioni, aiutando di fatto chiunque, indipendentemente dal livello di esperienza che si ha. Si possono trovare con facilità svariati utili e discussioni online e invece di iniziare da zero, è possibile accedere al codice di altri utenti e modificarlo per soddisfare le propie esigenze.

3.1.2 Docker

Docker è una piattaforma open source utilizzata per sviluppare, distribuire e gestire applicazioni in ambienti virtualizzati leggeri chiamati container. Si tratta di unità standardizzate che offrono tutto l'occorrente per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, senza che gli sviluppatori debbano preoccuparsi di problemi di compatibilità. Il confezionamento delle app in ambienti isolati (container) semplifica inoltre lo sviluppo, la distribuzione, la manutenzione e l'utilizzo delle applicazioni.

L'utilizzo di Docker consente di ottenere ed eseguire i suddetti container per eseguire un'ampia gamma di pacchetti software, tra cui anche KeyCloak.

3.1.3 GIT

Git è uno strumento per il versioning del progetto, utilizzabile da riga di comando. Nacque come alternativa a BitKeeper, dopo che venne revocata la licenza di uso gratuito da parte di chi ne deteneva i diritti d'autore. Le caratteristiche fondamentali del tool di versioning sono:

- *forte supporto allo sviluppo non lineare*: Git supporta diramazione e fusione, chiamate più comunemente branching e merging, rapide e comode, ed è comprensivo inoltre di strumenti specifici per visualizzare e navigare una cronologia di sviluppo non lineare;

- *sviluppo distribuito*: Git offre ad ogni sviluppatore la possibilità di avere una copia locale dell'intera cronologia di sviluppo;

- *I repository possono essere pubblicati facilmente tramite HTTP, FTP, ssh, rysinc o uno speciale protoccolo di git*;

- *gestione efficiente di grandi progetti*: Git risulta molto veloce e scalabile e più rapido di altri software di versioning presenti;

- *Autenticazione crittografica della cronologia*: la cronologia di Git viene memorizzata in modo tale che il nome di una revisione particolare, la cosiddetta "commit", dipenda dalla completa cronologia di sviluppo che conduce tale commit;

- *Strategie di fusione intercambiabili*: Git ha un modello ben definito per la gestione delle fusioni e ha diversi algoritmi per provare a completarla. Nel caso di fallimento, viene comunque proposta una fusione manuale da parte dell'utente.

Git permette e incoraggia la produzione di branch locali multipli, indipendenti l'uno con l'altro. La creazione, il merging e la cancellazione di questi avviene in pochi secondi. Questo permette cose come:

- *Frictionless Context Switching*: se allo sviluppatore salta in mente un'idea può crearsi un branch e lavorare su quello, poi tornare tranquillamente dal punto in cui era partito, applicare patch, tornare ancora nel branch sperimentale, fare merge del lavoro per aggiornare il sistema;

- *Role-Based Codelines*: si possono dotare i branch di un livello di importanza "virtuale". Ad esempio si può creare un branch dedicata al codice in produzione (master), un branch per lo sviluppo (develop), più tanti piccoli sotto branch per lo sviluppo giorno per giorno;

- *Feature based Workflow*: si possono creare tanti branch, ognuno per ogni caratteristica diversa che si vuole implementare, poi eliminarle quando la caratteristica viene applicata al codice in sviluppo grazie al processo di merge;

- *Disposable Experimentation*: con un branch si può sperimentare senza preoccuparsi di creare danni al codice in sviluppo. Se qualcosa non funziona basta tornare nel branch di sviluppo e scartare le modifiche.

Quando si carica il repository online, con il metodo di push, non si caricano tutti i branch, ma si può scegliere quali e quante caricarne. Questo porta a lasciare spazio alle persone, in particolare nello sviluppo in team, di provare le proprie idee senza la preoccupazione di minare e creare problemi ai processi creati e funzionanti in fase di sviluppo.

Il metodo di lavoro, il workflow, sul quale basare il proprio progetto è a scelta del programmatore o del team, non imposto da Git, e ne esistono di svariati. newline

Ci sono servizi legati a Git, uno tra questi è Bitbucket, una sorta di cloud di repository. Si andrà a legare Git al repository online, potendo per cui caricare i file, scaricarli, apportarne modifiche e clonarli.

3.2 Linguaggi utilizzati

3.2.1 Java

A livello generale, all'interno del contesto aziendale la scelta tecnologica di utilizzare Java come linguaggio di programmazione è abbastanza scontata: l'ambito Java è predominante e gode di un ampio supporto sul mercato.

Si è preferito quindi optare per una soluzione open source che garantisse un'ampia affidabilità e diffusione rispetto agli standard attuali di mercato.

Essendo Java un linguaggio gratuito e versatile, consente di realizzare sia software localizzati che distribuiti. Tra gli scenari di utilizzo comuni di Java sono inclusi lo sviluppo di videogiochi, il cloud computing, i big data, l'intelligenza artificiale e l'Internet of Things (IoT).

I vantaggi della scelta di Java come linguaggio di programmazione su cui basare uno o più progetti sono svariati:

- *risorse di apprendimento di alta qualità*, Java è presente sul mercato da diverso tempo, e per questo sono disponibili molte risorse di apprendimento per i nuovi programmatori. Documentazione dettagliata, testi completi e una considerevole quantità di corsi di ogni genere supportano gli sviluppatori che si approcciano alla programmazione Java;

- *funzioni e librerie integrate*, con l'utilizzo di Java, gli sviluppatori non devono scrivere ogni nuova funzione ex novo perchè avranno già a disposizione innumerevoli funzioni e librerie integrate per sviluppare le proprie applicazioni;

- *supporto attivo della community*, godendo di molti utenti attivi e di una forte e salda community, gli sviluppatori java hanno un'unlteriore potentissimo mezzo per affrontare le sfide del settore. Inoltre, il software della piattaforma Java viene regolarmente gestito e aggiornato;

- *strumenti di sviluppo di alta qualità*, Java offre vari strumenti a supporto di procedure come l'editing automatizzato, il debug, i test, l'implementazione e la gestione dei cambiamentoi. Tali strumenti rendono la programmazione con Java efficiente sia in termini di tempo, che di costi;

- *piattaforma indipendente*, il codice Java ha il vantaggio di poter essere eseguito su qualsiasi ambiente, che sia Windows, Linux, iOS o Android, senza riscriverlo. Ciò lo rende particolarmente efficiente sopratutto al giorno d'oggi, quando la tendenza è quella di voler eseguire applicazioni su più dispositivi;

- *sicurezza*, per quanto riguarda la sicurezza, gli utenti possono scaricare un codice Java non attendibile su una rete ed eseguirlo in un ambiente sicuro dove non può recare alcun danno. Tale codice non può infettare il sistema host con un virus, né può leggere o scrivere file dal disco rigido. I livelli di sicurezza e le restrizioni in Java sono altamente configurabili.

3.3 Panoramica del progetto

Il caso si studio preso come punto di partenza per la realizzazione di due diverse applicazioni per l'implementazione di un Single Sign On tra le stesse è descritto nel seguente modo:

"Supponiamo che il Ministero dell'Istruzione stia pianificando di creare un'integrazione Single Sign-On tra più scuole. L'idea di fondo è quella di mantenere gli ID degli account singoli dei propri studenti e insegnanti in più scuole, utilizzando una piattaforma centralizzata. Ciò consente a ciascun utente di avere lo stesso ruolo, ma con accesso e privilegi diversi in ogni scuola, a seconda dei ruoli."

3.4 Gestione utenze con Keycloak

Il primo passo per la realizzazione della nostra applicazione è assicurarsi di avere Docker installato e avviare Keycloak digitando il comando seguente da terminale:

```
docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:20.0.3 start-dev
```

Per impostazione predefinita, il suddetto comando estrae l'immagine della versione 20.0.3 del Keycloak e avvierà il container.

In questo modo viene configurato l'account amministratore sul comando docker run stesso utilizzando i parametri `KEYCLOAK_USER` e `KEYCLOAK_PASSWORD`, in questo caso adminadmin. Di default, Keycloak viene eseguito internamente sul container docker sulla porta 8080, ma è possibile modificare la porta host in base alle proprie esigenze.

Una volta eseguito il comando `docker run`, andando su `http:localhost:8080` si aprirà la pagina Keycloak.

A questo punto, aprendo la Keycloak Admin Console si potrà effettuare l'accesso come super admin inserendo le credenziali appena create e sarà possibile effettuare una serie di operazioni, tra cui la creazione di un **realm**.

Un realm in Keycloak è un'entità che permette di gestire un insieme di utenti con relative credenziali, ruoli e gruppi. In un server Keycloak si possono avere più realm che saranno completamente isolati e indipendenti gli uni dagli altri e ciascuno si occuperà della gestione dei propri utenti e applicazioni. E' quindi sufficiente utilizzare una singola installazione di Keycloak per molteplici scopi. Si potrebbe, per esempio, avere un realm per le applicazioni e i dipendenti interni e un altro per le applicazioni e i clienti esterni.

Di default Keycloak ha già un realm che prende il nome di "master", il cui unico scopo è creare e gestire altri realm nel sistema godendo dei privilegi amministrativi globali. Essendo al livello più alto nella gerarchia dei realm, gli account di ' amministrazione presenti nel master realm possono visualizzare e gestire qualsiasi altro realm creato nell'istanza del server.

Partendo dal presupposto che è necessario creare realm aggiuntivi per l'utilizzo basato sull'applicazione, creiamone uno dandogli il nome "Education". Successivamente, creiamo anche un primo user settando nome utente e password (non temporanea).

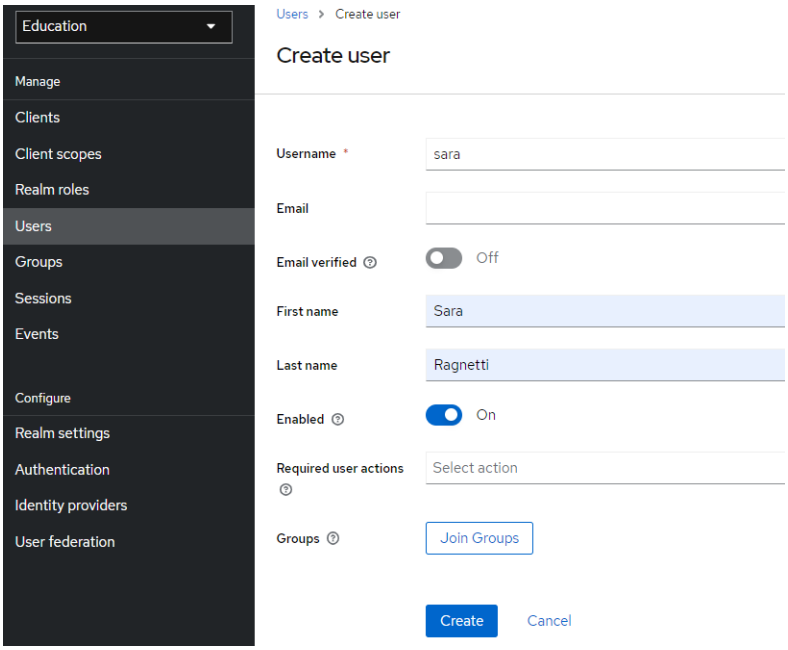


Figura 3.1: Creazione di uno user.

Accedendo ora alla Keycloak Account Console con *sara* e la password settata si potrà gestire tale account. In particolare, la console fornisce un'interfaccia da cui gli utenti gestiscono in modo centralizzato i propri account, in particolare possono aggiornare il proprio profilo utente, la password, oppure abilitare l'autenticazione a due fattori o visualizzare le applicazioni, incluse quelle a cui

si sono autenticati. Gli consente anche di visualizzare le sessioni attive ed effettuare la disconnessione remota da altre sessioni. Inoltre, se il social login o l'identity brokering sono abilitati, un utente può anche collegare i propri account a provider aggiuntivi. E possibile accedere alla Keycloak Account Console tramite il seguente URL relativo: `/auth/realms/realm-name/account` ed il suo aspetto viene mostrato nella figura seguente.

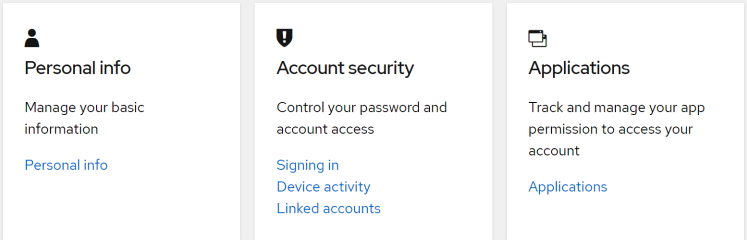


Figura 3.2: Inetrfaccia della Keycloak Account Console.

Per realizzare un sistema Single Sign On che differenzi gli accessi e riconosca gli utenti in base al ruolo occorre aggiungere due ruoli nella sezione dedicata all'interno del realm. Le applicazioni spesso assegnano autorizzazioni e permessi a ruoli specifici anzichè a singoli utenti dato che la gestione degli utenti potrebbe essere troppo complessa. Nel nostro caso ci basterà inserire i seguenti ruoli:

- `student`;
- `teacher`;

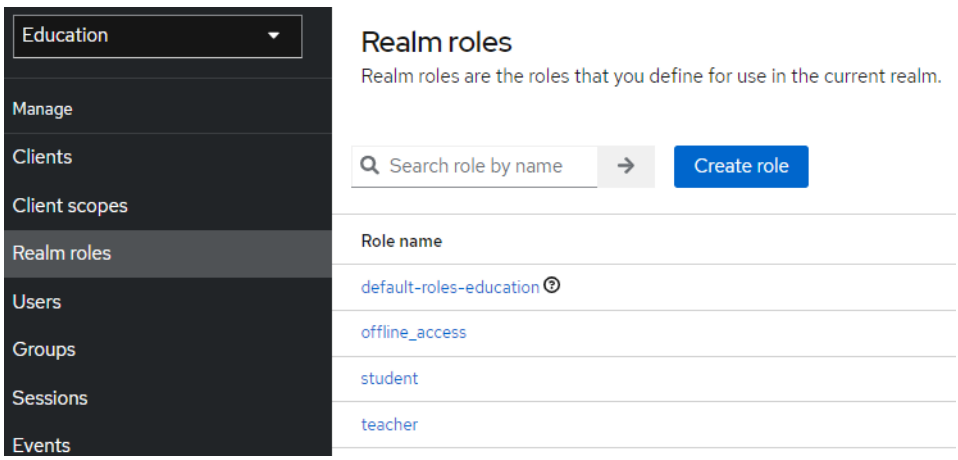


Figura 3.3: Lista dei ruoli del realm Education.

Ogni realm può essere composto da più **client**. Un client Keycloak rappresenta un'applicazione o un servizio Web per autenticare e autorizzare gli utenti.

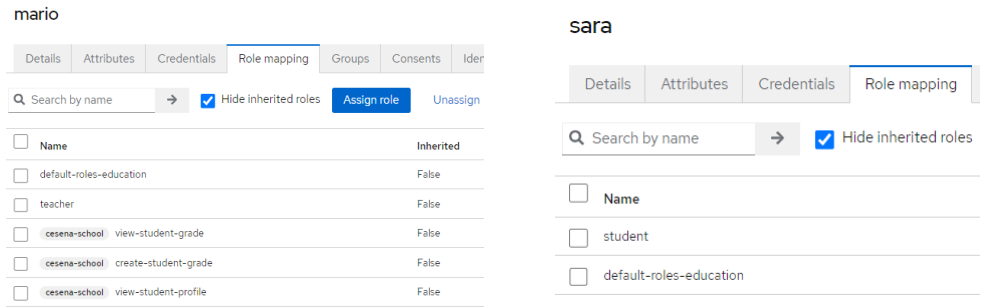
Keycloak supporta sia OpenID Connect (un'estensione di OAuth 2.0) che SAML 2.0. Il primo step per la gestione della sicurezza di client e servizi è decidere quale dei due protocolli utilizzare. Creiamo quindi un client denominato `cesena-school` specificando la scelta di `OpenID Connect`.

Un client è un'applicazione o un servizio che intende utilizzare Keycloak per fornire il servizio di Single Sign-On. Potrebbe anche essere un'entità che richiede di avere accesso a nome utente e password dell'utente in modo da scambiarli direttamente con il server per ottenere l'Access Token (token di accesso) e invocare altri servizi che utilizzano Keycloak. Questo è possibile solo abilitando, durante la configurazione, il *Direct Access Grant*.

Oltre ai ruoli a livello di realm, possiamo assegnare quelli specifici all'interno di ciascun client. Andiamo quindi a creare i seguenti ruoli per specializzare ulteriormente gli utenti:

- `create-student-grade`;
- `view-student-grade`;
- `view-student-profile`.

Una volta creati tutti i ruoli che ci interessano occorre assegnarli ai vari utenti. Tali informazioni saranno incapsulate in token e assertion in modo che le applicazioni possano definire i permessi di accesso alle proprie risorse. Nel nostro caso avremo la seguente mappatura utente-ruolo:



(a) Ruoli user mario (teacher). (b) Ruoli user sara (student).

Figura 3.4: Risultato dell'assegnamento dei rispettivi ruoli agli utenti.

Quando viene creato un Access Token OIDC o un'assertion SAML, tutte le mappature dei ruoli utente dell'utente vengono automaticamente aggiunte

come claims all'interno del token o dell'assertion. Le applicazioni utilizzano poi queste informazioni per regolare l'accesso alle risorse.

In Keycloak, gli access token sono firmati digitalmente e possono essere effettivamente riutilizzati dall'applicazione per invocare altri servizi REST protetti da remoto. Ciò significa che se un'applicazione venisse compromessa o se fosse presente un client non autorizzato registrato con il realm, i malintenzioanti potrebbero ottenere degli access token con un'ampia gamma di autorizzazioni e l'intera rete verrebbe messa a rischio.

Per prevenire questo problema, Keycloak ha messo a disposizione i cosiddetti **client scopes**. Si tratta di entità configurate a livello di realm che possono essere collegate ai client. Vengono fatti dei riferimenti per nome ai client scope quando viene inviata una richiesta all'endpoint Keycloak di autorizzazione.

In questo modo si limitano i ruoli che vengono dichiarati all'interno degli access token. Così, quando un client richiede l'autenticazione di un utente, l'access token che riceve in risposta conterrà solo le mappature dei ruoli specificate esplicitamente per quel client scope.

I client scopes assegnati di default agli utenti del realm Education sono `roles` e `profile`.

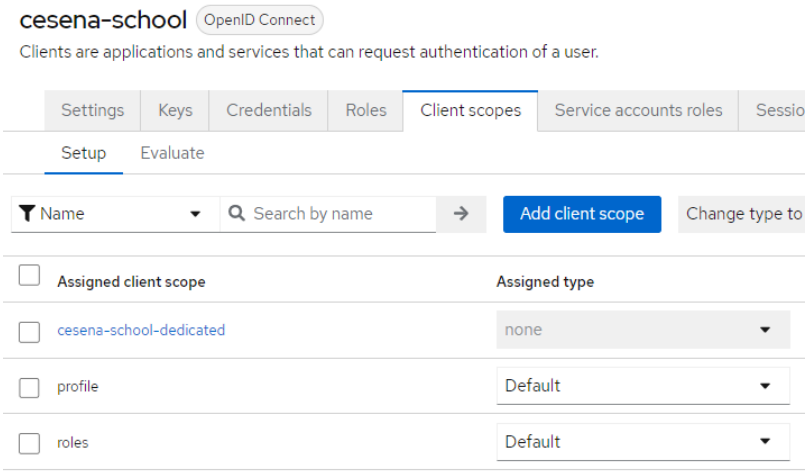


Figura 3.5: Lista dei client scopes assegnati di default.

In Keycloak, la funzione dei protocol mapper è quella di aggiungere ulteriori claims ad un JWT, oltre a quelli già presenti per impostazione predefinita.

Per ciascun Protocol Mapper ci sono opzioni diverse, ovvero è possibile specificare se aggiungere o meno alcune informazioni in:

- ID token;
- access token;

- user info.

Ciò che accadrà è che Keycloak crea il JWT (un oggetto JSON codificato basato su uno standard specifico), con i claims registrati (i.e. Issuer, Subject e così via), quindi applicherà anche i claims personalizzati (ovvero i protocol Mapper) dall'ordine di priorità che è stato definito per quel dato mapper.

Il token che ne risulta sarà un JWT composto da claims registrati predefiniti e claims privati, aggiunte con i protocol mapper.

Settiamo i mappers per fare in modo che i ruoli siano visibili nella userinfo API.

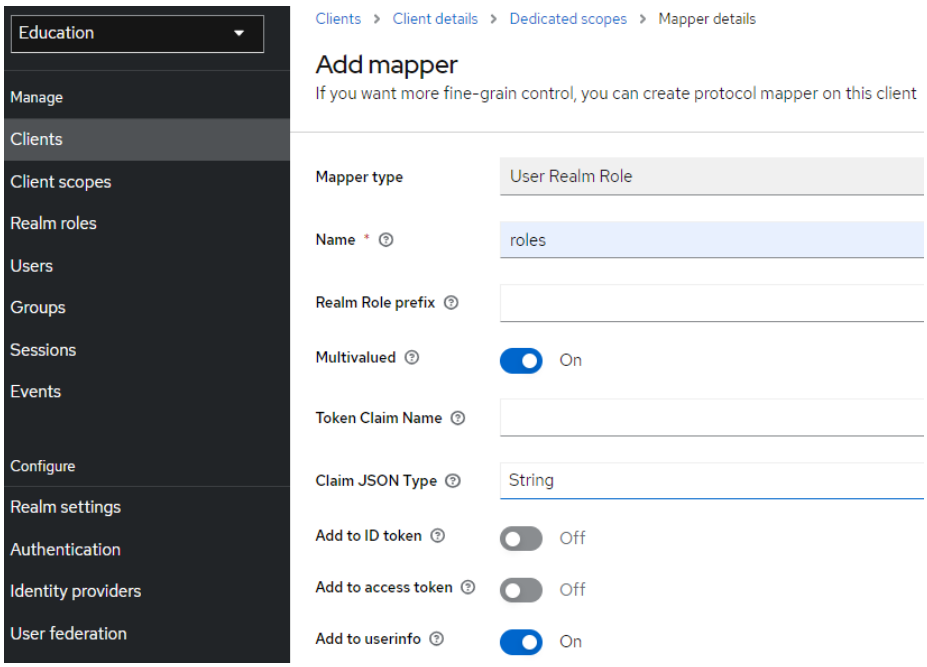


Figura 3.6: Lista dei client scopes assegnati di default.

3.5 Social Login in Keycloak

Keycloak può essere configurato per delegare l'autenticazione a uno o più Identity Provider. Il social login tramite Facebook o Google+ ne è un esempio. In alternativa, si può anche collegare Keycloak a qualsiasi altro IDP OpenID Connect o SAML 2.0 (Twitter, GitHub, LinkedIn, Microsoft, Stack Overflow, ecc).

Vediamo, a titolo d'esempio, come funziona il social login con Github su Keycloak. Innanzitutto riassumiamo brevemente le fasi del flusso dell'identità:

- per prima cosa l'utente tenta di accedere ad una risorsa (applicazione). Se non si è ancora autenticato viene ridirezionato alla pagina di login dell'Identity broker (Keycloak);
- l'utente può autenticarsi sfruttando il login di Keycloak, oppure può utilizzare il bottone di social login, in questo caso quello di Github;
- l'identity broker scambierà il token di autorizzazione dal provider di identità, che chiederà all'utente di approvare le autorizzazioni;
- successivamente, dopo le approvazioni da parte dell'utente, l'identity broker aprirà una sessione di autenticazione per l'utente, che otterrà l'accesso alla risorsa per cui ha fatto richiesta.

Per poter fare social login per la nostra applicazione con Keycloak vanno fatte varie operazioni sia lato Github che lato Keycloak.

3.5.1 configurazioni Github

La prima cosa da fare è creare una nuova Oauth App dall'apposita sezione `https:github.comsettingsdevelopers` . Una volta creata, sarà possibile visualizzare il `client id` e il `client secret` generati per la nostra applicazione.

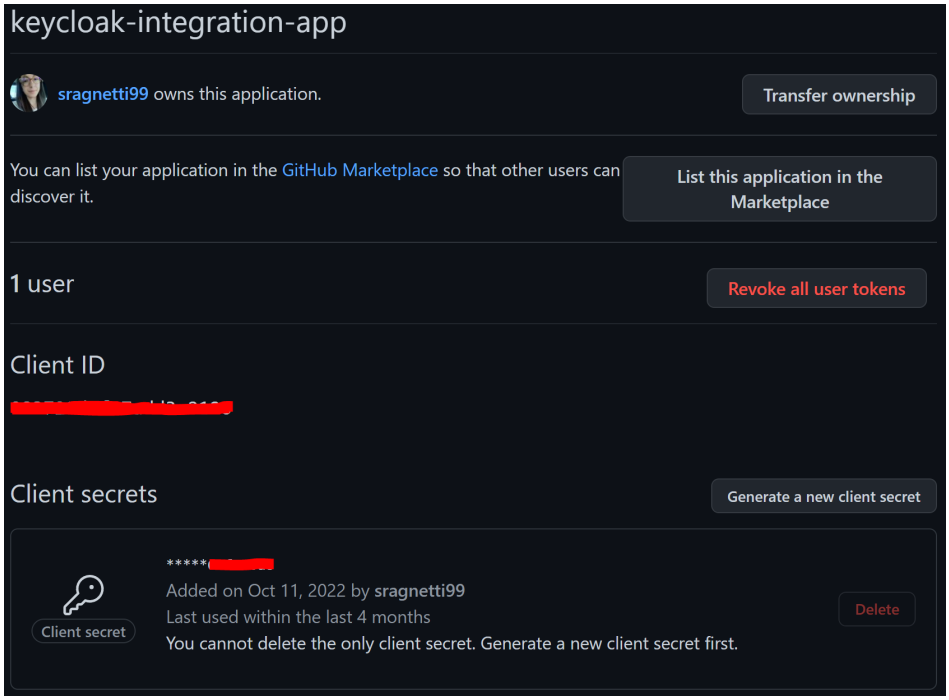


Figura 3.7: Oauth App appena creata.

3.5.2 Configurazioni Keycloak

Lato Keycloak, invece, andando nella sezione "Identity Providers", bisogna aggungere un nuovo IdP Github e inserire il `client id` e il `client secret`.

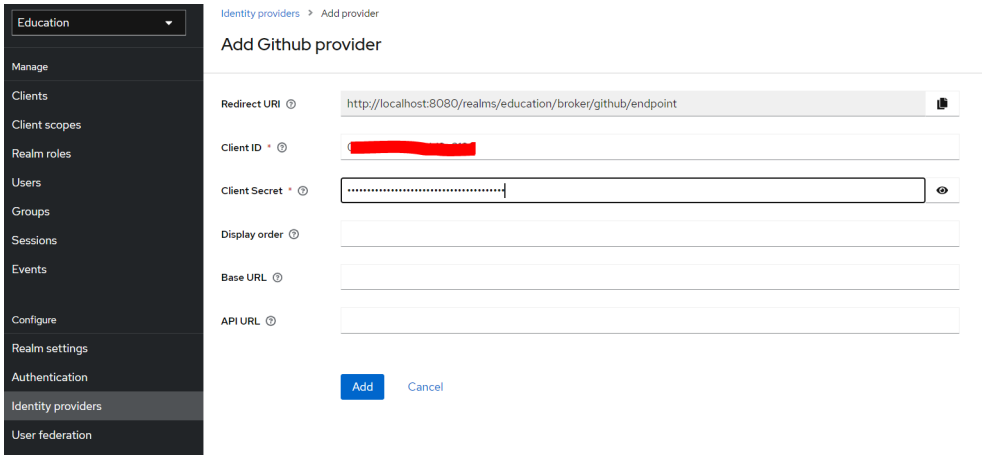


Figura 3.8: Github provider.

Per effettuare il social login per la nostra app basta andare all'indirizzo `http://localhost:8080/realms/education/account/#/` e cliccare sul bottone "sign in with...Keycloak".

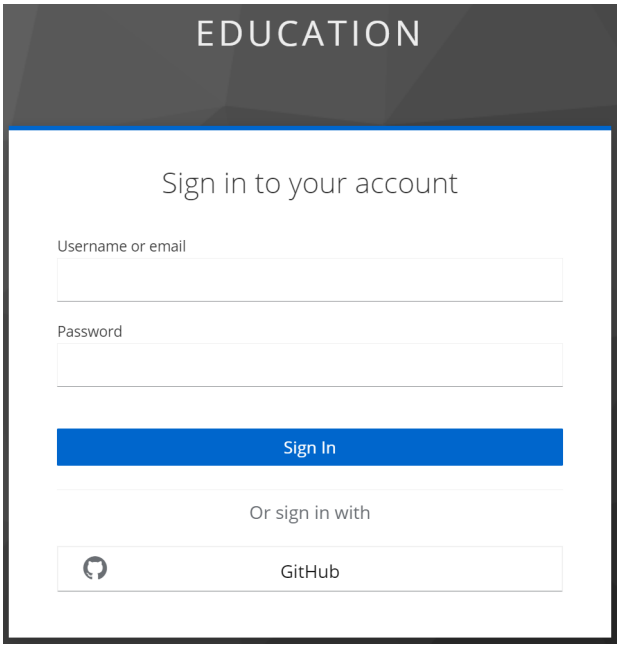


Figura 3.9: Schermata del social login.

3.6 Applicazione 1

Entrambe le applicazioni realizzate si connettono alle istanze di Keycloak e utilizzano la funzionalità di autenticazione e autorizzazione del server tramite le proprie API REST, ma con alcune differenze.

A grandi linee, possiamo dire che la prima applicazione consente ai vari utenti di effettuare il login e il logout attivando e disattivando le rispettive sessioni sul server Keycloak e di visualizzare informazioni diverse se, in base al ruolo ricavato attraverso il JWT, ne hanno il permesso. La seconda applicazione, invece, sfruttando l'access token generato dall'altra app, è in grado di riconoscere tutti gli utenti e i rispettivi ruoli.

Per sviluppare un'applicazione che utilizzi Spring è fondamentale scaricare un template per il progetto tramite Spring Initializr [19], dopo averlo opportunamente configurato. In particolare, selezioniamo Maven come gestore delle dipendenze, Java come linguaggio da utilizzare e aggiungiamo alcune dipendenze fondamentali:

- `Lombok`, una libreria di tipo Annotation processor (APT) che durante la compilazione del progetto esegue l'interpretazione delle cosiddette `annotation` dichiarate a livello di classe;
- `Spring Web`, una libreria che per la creazione di applicazioni Web, incluse le RESTful, utilizzando Spring MVC;
- `Spring Security`, framework che fornisce autenticazione, autorizzazione e altre funzionalità di sicurezza delle applicazioni Java ed è, di fatto, lo standard per la sicurezza di applicazioni basate su Spring.

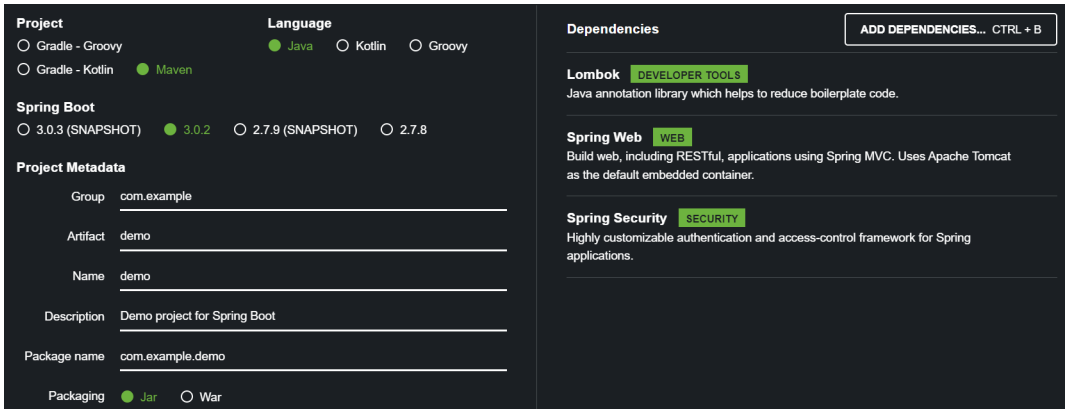


Figura 3.10: Interfaccia per la creazione di un template Spring.

3.6.1 Properties file

Per connettere l'applicazione al server Keycloak precedentemente predisposto occorre configurare opportunamente uno specifico file properties, inserendo manualmente informazioni quali *client-id*, *client-secret* e i vari URI.

```
server.error.whitelabel.enabled=false
spring.mvc.favicon.enabled=false
server.port = 8083
keycloak.client-id=cesena-school
keycloak.client-secret=
keycloak.scope=profile
keycloak.authorization-grant-type=password
keycloak.authorization-uri=http://localhost:8080/realms/education/protocol/openid-connect/auth
keycloak.user-info-uri=http://localhost:8080/realms/education/protocol/openid-connect/userinfo
keycloak.token-uri=http://localhost:8080/realms/education/protocol/openid-connect/token
keycloak.logout=http://localhost:8080/realms/education/protocol/openid-connect/logout
keycloak.jwk-set-uri=http://localhost:8080/realms/education/protocol/openid-connect/certs
keycloak.certs-id=2Wfn9Bmi4riAQM7Nq7tWRDIpwTRRtO7IBvLxwGcCXN4
logging.level.root=INFO
```

Figura 3.11: Snapshot del codice contenuto nel file di properties.

La classe `RestTemplateConfig` consente all'applicazione di leggere infomazioni dal file properties grazie all'annotazione `PropertySource` a cui viene dato come `value` il nome del file stesso:

`@PropertySource(value = "classpath:application.properties")`

La suddetta classe ha anche un'altra annotazione, `@Configuration`, che indica che la classe contiene dei metodi annotati con `@Bean`. Un Bean è un componente che "vive" solo all'interno di uno specifico contesto e che può essere iniettato (con `@Autowired` o `@Inject`) a seconda del bisogno.

Il questo modo Spring può elaborare la classe e generare dei beans Spring da utilizzare nell'applicazione. Il `Bean` in questione è un `RestTamplate`, un client sincrono per eseguire richieste HTTP, esponendo una semplice template method API su librerie client HTTP sottostanti.

3.6.2 Il package Service

Nella maggior parte delle applicazioni, si hanno livelli distinti, come l'accesso ai dati, la presentazione, il servizio, la logica di business, ecc. Ogni livello contiene a sua volta dei Beans. Per rilevare automaticamente questi Bean, Spring effettua il classpath scanning delle annotazioni, registrando ciascun Bean nell'`ApplicationContext`.

L'annotazione `@Configuration` discussa nel paragrafo precendente è solo una delle varie annotazioni di cui Spring si serve per individuare facilmente i

diversi componenti. Molto importante è infatti l'annotazione `@Service`, che viene aggiunta in tutte quelle classi che implementano la logica per archiviare, recuperare, eliminare e aggiornare dati e risorse.

Lo Spring context rileverà automaticamente le classi con questa annotazione attraverso il components scan, dato che `@Service` è una specializzazione di `@Component`.

Classe JwtService

La classe `JwtService` contiene due metodi che vengono utilizzati dal Controller e all'interno delle classi di Test per verificare la validità dei Token. Entrambi i metodi hanno la medesima funzionalità, ovvero quella di restituire la Json Web Key sfruttando i certs-id e l'url, ma con differente campo di applicazione (cioè con una diversa lista di parametri).

Più nello specifico, il metodo `getJwk()` restituisce la JWK relativa al Key id "`certsId`" mediante il Provider che punta all'URL specificato "`jwksUrl`" per caricare il set di JWK.

```
public Jwk getJwk() throws Exception {
    return new UrlJwkProvider(new URL(jwksUrl)).get(certsId);
}
```

Figura 3.12: Snapshot del codice relativo al metodo `getJwk()`.

Per specificare i valori da assegnare ai campi `certsId` e di `jwksUrl` si utilizza l'annotazione `@Value` nella sua forma più comune, ovvero quella in cui il valore specificato tra parentesi è il placeholder di una proprietà specificata in un file di proprietà:

```
1 usage
@Value("${keycloak.jwk-set-uri}")
private String jwksUrl;

1 usage
@Value("${keycloak.certs-id}")
private String certsId;
```

Figura 3.13: Snapshot del codice relativo ai campi della classe `JwtService`.

Classe KeycloakRestService

La classe `KeycloakRestService` è la classe che contiene le vere request HTTP, comunicando di fatto con il server Keycloak. I suoi metodi, infatti, ve-

nogno chiamati all'interno delle API per effettuare operazioni di login, logout, richiesta di informazioni, e verifica della validità dei token.

Per fare ciò occorre avere un campo di tipo `RestTemplate`, opportunamente annotato con `@Autowired` in modo che venga identificato automaticamente e che gli venga anche assegnato un valore che soddisfi la dipendenza.

L'Autowiring è, infatti, il processo per mezzo del quale Spring definisce le dipendenze tra beans, sollevando lo sviluppatore da doverle definire esplicitamente. A tale scopo si annotando campi e costruttori o metodi per indicare al framework le dipendenze del bean.

Anche in `KeycloakRestService` ci sono diversi campi annotati con `@Values` e valorizzati con le relative informazioni del file di properties.

Analizziamo nel dettaglio i metodi della classe `KeycloakRestService`:

- `login(String username, String password)`, metodo che permette di effettuare il login su keycoak utilizzando le credenziali ("`username`" e "`password`") passate come argomenti. Per creare la request HTTP si devono aggiungere coppie chiave-valore di username, password, client id, grant type, client secret e scope ad una mappa che andrà a formare il body della request. Creata la request, si utilizzerà il metodo `postForObject()` di `RestTemplate` che crea una nuova risorsa eseguendo il POSTing dell'oggetto specificato nell'URI e restituisce la rappresentazione trovata nella response. In questo caso, verrà restituito il token all'interno del body;

- `logout(String refreshToken)`, metodo per effettuare il lougout dell'utente che ha come effetto la disconnessione della sessione dal server Keycloak e la conseguente disattivazione del token. A differenza di altre request, in questo caso viene utilizzato il `Refresh Token` e non il solito `Access Token`. Anche qua si sfrutta il metodo `postForObject()` e verrà restituito un codice HTTP a seconda dell'esito della richiesta;

- `getUserInfo(String token)`, metodo per ottenere informazioni sull'utente e che, allo stesso tempo, verifica la validità del `token`. Viene creata una nuova risorsa effettuando il POSTing dell'oggetto dato verso l'URI specificato e si restituisce la rappresentazione trovata di tale oggetto.

3.6.3 Il package Controller

I Controller sono una parte fondamentale del paradigma MVC ed hanno il compito di rispondere alle Request, generando delle Response. A prescindere da come sia stata inviata una richiesta, la risposta ad essa associata sarà sempre generata da un Controller, almeno in una Web Application creata con Spring.

In altre parole, il Controller è l'entry point, ovvero il "punto di ingresso" di un'applicazione web. Un'applicazione può avere più entry point diversi, ognuno relativo ad una specifica funzionalità.

Il Controller non è altro che una classe Java che implementa i metodi di un'interfaccia al cui nome viene aggiunta la parola "`Api`", ciascuno dei quali è associato ad un diverso indirizzo URL. Quando arriva una request verso quell'indirizzo verrà eseguito il metodo corrispondente.

AuthenticationApi

Opportunamente annotata con il tag `@Api`, `AuthenticationApi` è l'interfaccia che espone i metodi dell'API Rest, definendo, per ciascuno, una descrizione dell'operazione e un wrapper per una lista di più oggetti di tipo `ApiResponse`. Nella figura seguente è possibile vederne un esempio.

```
@ApiOperation(value = "Effettua il login dell'utente")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "Ok", response = ResponseMessage.class),
        @ApiResponse(code = 400, message = "L'utente non esiste", response = String.class)
})
String login(String username, String password);
```

Figura 3.14: Dichiarazione del metodo `login()`.

Avendo a disposizione svariati codici di stato HTTP, ne sono stati individuati e quindi utilizzati solo una parte per le request della nostra API.

- `200 OK`, response standard per le richieste HTTP andate a buon fine;
- `400 Bad Request`, in generale è la risposta che si riceve quando un errore di sintassi non consente di soddisfare le request. In questo caso fa riferimento alla non validità del token oppure ad un utente inesistente;
- `403 Forbidden`, nonostante la legittimità della request, il server si rifiuta di soddisfarla. Si verifica quando, ad esempio, viene negato il permesso per l'URL richiesto.

Di questi metodi si parlerà nel dettaglio nel sottoparagrafo seguente.

AuthenticationController

La classe `AuthenticationController` è l'unico controller dell'applicazione ed è annotato con:

- `@RestController`, annotazione utilizzata a livello di classe che consente alla stessa di gestire le richieste effettuate dal client, permettendo di fatto di definire un'API REST. All'interno di un servizio web RESTful, i controller prendono questa denominazione speciale;
- `@RequestMapping("/api/v1/auth")`, serve per configurare una mappatura per le richieste esterne. Associa ai metodi della classe una URL per le richieste. Se utilizzata a livello di classe, il valore (in questo caso texttt/api/v1/auth) viene applicato a tutti i metodi della classe, ma ogni metodo può prevedere una porzione aggiuntiva per il path della URL mediante l'annotazione `@GetMapping(value="/percorso")` oppure `@PostMapping(value="/percorso")`, a seconda dei casi.

Questa classe sfrutta i metodi dei due Service per soddisfare le request generate mediante un apposito tool, vale a dire Postman. Si tratta di uno strumento capace di inviare request a qualsiasi server e di ricevere le relative response. Per poter utilizzare Postman per inviare le richieste correttamente occorre avviare sia l'applicazione dall'IDE, sia il container docker associato.

Login

Per effettuare il login dell'utente e attivare la relativa sessione sul server Keycloak occorre selezionare, per prima cosa, il metodo POST e indicare la URL corretta, che è composta da una parte relativa alla porta su cui è esposto il server (`http://localhost:8081`), una relativa all'API (`/api/v1/auth`) e una specifica del metodo per il login (`/login`). E questo vale per tutte le request di cui si discuterà in seguito. L'unica differenza sarà, appunto, la porzione di path relativa al metodo.

Per le credenziali si utilizzano quelle dell'utente creato precedentemente.

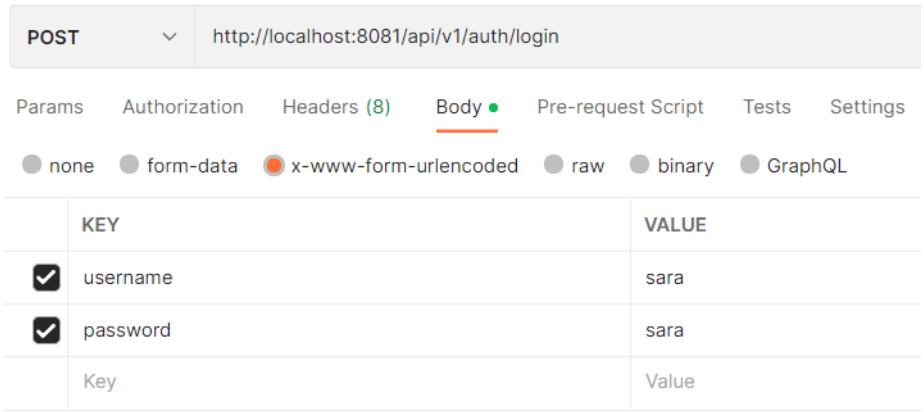


Figura 3.15: Login request via Postman.

In caso di successo (l'utente esiste e le credenziali sono corrette) il body della response conterrà una serie di informazioni sull'utente, tra cui, ovviamente, il suo access token, ovvero quel token che dovrà essere riconosciuto ed utilizzato per effettuare ulteriori request dalla seconda applicazione. Se, al contrario, le credenziali inserite non corrispondono a quelle di un utente registrato, la response conterrà, nel body, la seguente stringa:

```
{
    "success": false,
    "code": "NON_EXISTENT_USER",
    "message": "L'utente inserito non esiste"
}
```

Listato 3.1: Body della response di una request (login) non andata a buon fine.

```
{
    "access_token":
        ↪ "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6I...",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token":
        ↪ "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6...",
    "token_type": "Bearer",
    "not-before-policy": 0,
    "session_state": "deabd6f5-3465-418f-92d1-fecb785597e1",
    "scope": "profile"
}
```

Listato 3.2: Body della response di una request (login) non andata a buon fine.

Oltre all'access token, è possibile visualizzare anche il refresh token e la loro durata espressa in secondi, lo scope e il tipo di token, in questo caso "Bearer". Tale tipologia di token è quella usata per ottenere l'autorizzazione ad accedere ad una risorsa protetta da un Authorization Server conforme con lo standard OAuth2.

Di seguito è riportata la schermata delle sessioni attive su Keycloak. E' correttamente presente l'accesso dell'utente *sara*.

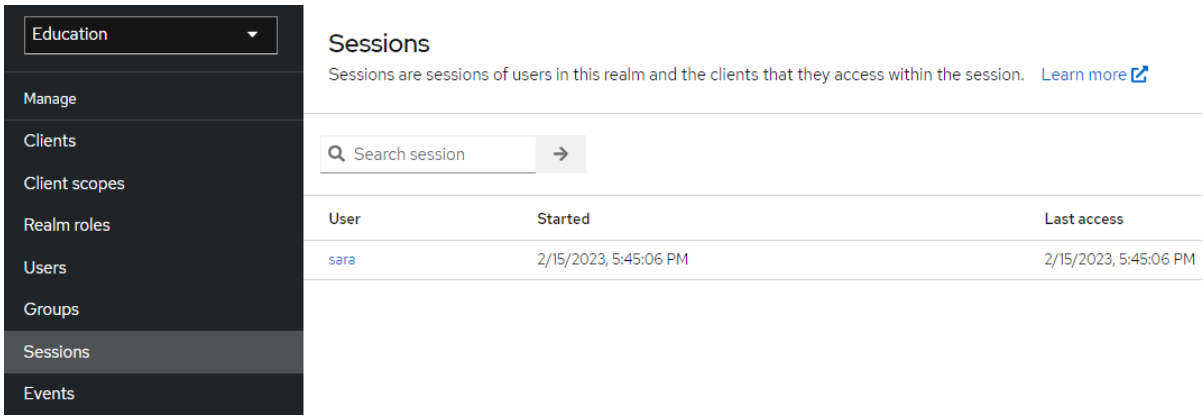


Figura 3.16: Sessioni attive sul server Keycloak.

GetStudentRoles

Il metodo `GetStudentRoles` prevede un unico argomento, ovvero l'access token dell'utente che sta effettuando la request. Questo token andrà inserito tra gli header della richiesta e sarà preceduto dalla stringa `"Bearer "`, in corrispondenza della key `Authorization`.

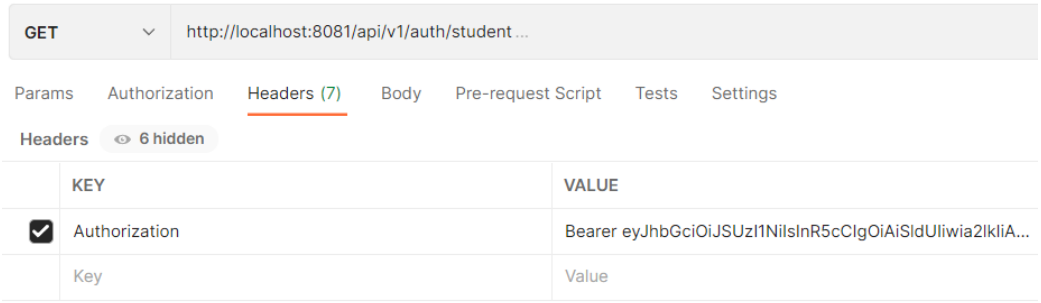


Figura 3.17: GetStudentRoles request via Postman.

Questo metodo va a leggere il token inserito allo scopo di estrapolare da esso alcune informazioni circa l'utente che ha effettuato la richiesta. Se l'utente è uno studente, significa che ha i permessi di richiedere determinate informazioni, che in questo caso equivalgono alla lista di ruoli che ha all'interno dell'organizzazione, e gli verranno quindi fornite attraverso la reponse.

Per fare ciò, prima di tutto viene decodificato il token, poi si verifica che sia ancora valido e che non sia scaduto quindi si controlla infine se tra la lista di ruoli all'interno del claim *roles* sia presente il ruolo *student*.

A seconda dei casi, possono essere generati diversi codici di stato HTTP a seguito della request. `200` se l'utente è uno studente e quindi la response

conterrà correttamente la lista dei suoi ruoli; `400` se il token non è valido, vale a dire se c'è un errore nella sintassi oppure se il token scaduto); `403` se l'utente non ha il ruolo di studente e quindi gli viene negato l'accesso alla risorsa.

```
[
    "student",
    "default-roles-education",
    "offline_access",
    "uma_authorization"
]
```

Listato 3.3: Body della response di una richiesta (GetStudentRoles) andata a buon fine.

GetTeacherRoles

Per il metodo `GetTeacherRoles` vale quanto detto per `GetStudentRoles`. Infatti, anche qua è previsto l'inserimento dell'access token nell'header della richiesta e i codici di stato delle reponse funzionano allo stesso modo.

L'unica differenza, chiaramente, risiede nel fatto che l'esito della request sarà positivo solo se l'utente coinvolto assume, all'interno dell'organizzazione, il ruolo di *teacher*.

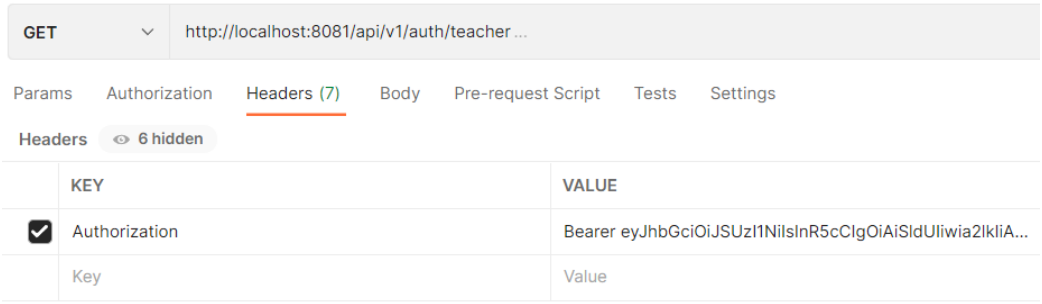


Figura 3.18: GetTeacherRoles request via Postman.

```
[
    "teacher",
    "default-roles-education",
    "offline_access",
    "uma_authorization"
]
```

Listato 3.4: Body della response di una richiesta (GetTeacherRoles) andata a buon fine.

GetUserInfo

Anche in `GetUserInfo` viene richiesto il Bearer token nell'header. Lo scopo del metodo è di restituire alcune informazioni sull'utente, a prescindere dal suo ruolo. Infatti, la request andrà a buon fine sia se l'utente coinvolto è student, sia se è teacher. `GetUserInfo` sfrutta il metodo omonimo definito nella classe `KeycloakRestService` discusso precedentemente nel paragrafo 3.6.2.

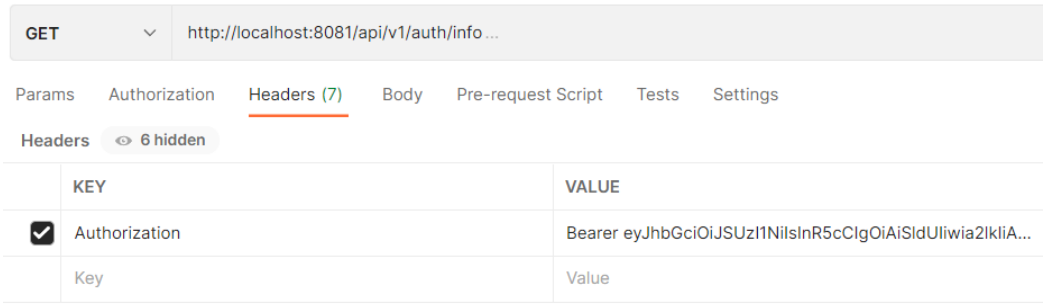


Figura 3.19: GetUserInfo request via Postman.

Tra le informazioni restituite è possibile visualizzare nome, cognome e username dell'utente e soprattutto la lista dei suoi ruoli.

```
{
    "sub": "db3c8a14-52d0-41b4-8199-352efdc262cb",
    "roles": [
        "student",
        "default-roles-education",
        "offline_access",
        "uma_authorization"
    ],
    "name": "Sara R",
    "preferred_username": "sara",
    "given_name": "Sara",
    "family_name": "R"
}
```

Listato 3.5: Body della response di una request (GetUserInfo) andata a buon fine.

CheckTokenValidity

Il metodo `CheckTokenValidity` verifica la correttezza sintattica del token e la sua validità. Se il token è corretto e non è ancora scaduto restituisce un

codice di stato HTTP 200, 400 in caso contrario.

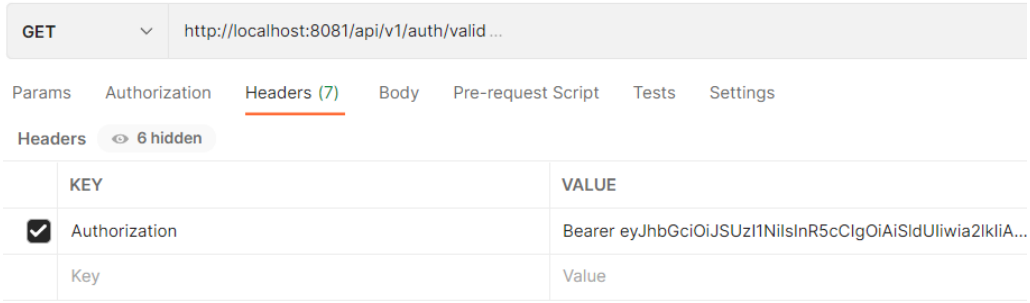


Figura 3.20: CheckTokenValidity request via Postman.

Logout

`logout` è l'ultimo metodo del controller `AuthenticationController`. A differenza degli altri, necessita del `refresh_token`.

Il logout viene utilizzato per disconnettere l'utente dalla sessione di single sign-on su Keycloak, disabilitando sia il refresh token che l'access token. Eventuali altri token attivi per l'utente dovranno invece essere mantenuti validi. Il refresh token viene revocato solamente se la richiesta contiene il cookie ad esseo associato oppure se il refresh token è stato inserito come `request parameter`.

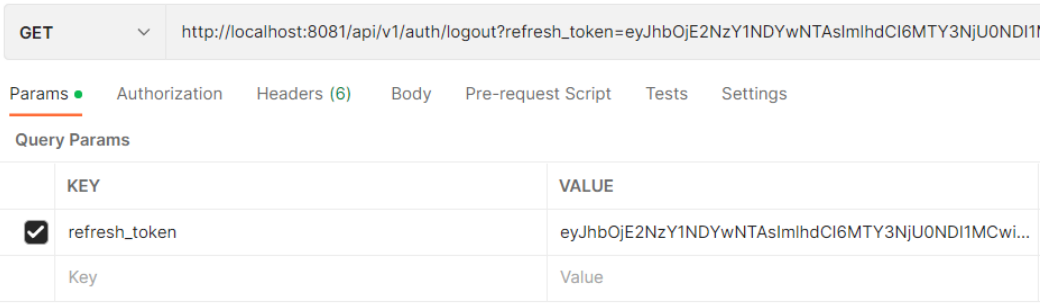


Figura 3.21: Logout request via Postman.

3.6.4 Test

Per testare il corretto funzionamento dell'applicazione è stata realizzata una semplice classe di Test, denominata `AuthenticationTest`. Come prima cosa viene eseguito un metodo chiamato `generateToken()` in cui si costruisce il token a partire da una serie di informazioni essenziali (username, password, client_id, grant_type, client_secret e scope). Poi lo si decodifica per verificarne

la correttezza. Tale token verrà utilizzato in tutti i seguenti metodi per ulteriori controlli, motivo per cui è annotato con `@BeforeAll`.

```
@BeforeAll
static void generateToken() throws Exception {
    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add( k: "username", v: "sara");
    map.add( k: "password", v: "sara");
    map.add( k: "client_id", v: "cesena-school");
    map.add( k: "grant_type", v: "password");
    map.add( k: "client_secret", v: "0FBdh4457iYE1VknB8EynvgAy1Xn5iko");
    map.add( k: "scope", v: "openid");
    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, new HttpHeaders());
    String result = restTemplate.postForObject(
            url: "http://localhost:8080/realms/education/protocol/openid-connect/token",
            request,
            String.class);
    token = new JSONObject(result).get("access_token").toString();
    Jwk jwk = jwtService.getJwk(JWSK_URL, CERTS_ID);
    decodedJWT = JWT.require(Algorithm.RSA256((RSAPublicKey) jwk.getPublicKey(), privateKey: null)) Verification
            .build() JWTVerifier
            .verify(token);
}
```

Figura 3.22: Metodo `generateToken()` per il testing.

I metodi successivi, invece, verificano la correttezza dei claims e dei valori specificati nel payload del token.

3.7 Applicazione 2

Lo scopo della seconda applicazione è quello di mostrare il funzionamento effettivo del single sign-on per cui si sono poste le basi grazie alle precedenti configurazioni e progettazioni.

Le due diverse app hanno le classi `@Service` pressochè identiche, con l'unica differenza che la seconda contiene, all'interno di `KeycloakRestService`, soltanto il metodo per effettuare il login.

3.7.1 Package Model e resources

Per semplificare e alleggerire il carico di lavoro, non essendo tra gli obiettivi di questo progetto l'integrazione con una base di dati, sono stati creati due file.json `esami.json` e `esamiByStudenti.json` che contengono, rispettivamente, la lista di esami da prenotare e l'elenco degli esami prenotati da ciascuno studente.

Exam

Dovendo lavorare con dei file.json in un'applet Java, occorre creare una classe di modello che rappresenti gli oggetti `Exam` e che permetta la loro serializzazione e deserializzazione.

```
public class Exam {
    1 usage
    @JsonProperty("codice")
    private String codice;
    1 usage
    @JsonProperty("data")
    private String data;
    1 usage
    @JsonProperty("materia")
    private String materia;

    sara ragnetti
    public Exam() {}

    sara ragnetti
    @Override
    public String toString() {...}

}
```

Figura 3.23: Classe `Exam`.

3.7.2 Package Controller

AuthenticationApi

Per l'API vale quanto detto nel paragrafo 3.6.3. In questo caso però verrà utilizzato un ulteriore codice di stato HTTP, ovvero `500 Internal Server Error`, che indica che il token è scaduto.

AuthenticationController

Il metodo per il login è stato aggiunto solo a scopo dimostrativo, per far vedere cioè che è possibile loggarsi direttamente anche tramite questa applicazione, anche se per svolgere tutte le altre operazioni non sarà affatto necessario.

`getExamsRegistration()` e `getAllExams()`, infatti, saranno in grando di riconoscere l'utente tramite il suo JWT e di stabilire di conseguenza chi ha accesso ad una determinata risorsa e chi no.

GetExamsRegistration

Il metodo `getExamsRegistration`, sfruttando il token generato opportunamente dall'applicazione 1, è in grado di riconoscere l'utente e di interpretare correttamente il suo ruolo, stabilendo di fatto se questo ha il permesso di accedere alla risorsa per cui fa richiesta oppure no.

Se l'utente che invia la request è uno `student`, verranno mostrati tutti gli esami da lui prenotati.

Con il metodo `getClaim()` applicato al token decodificato `jwt` si vanno a prendere tutti i ruoli dell'utente in corrispondenza del claim `realm_access`. Poi si controlla che nella lista di ruoli sia presente `student`. In caso affermativo, si procede con l'estrazione dell'username dell'utente dal token che verrà utilizzato come chiave per la ricerca di tutti gli esami da lui prenotati indicati nel file `esamiByStudent.json` in corrispondenza del valore esami_prenotati.

```
@GetMapping(path="/esamiPrenotati",produces = MediaType.APPLICATION_JSON_VALUE)
public List<String> getExamsRegistration(@RequestHeader("Authorization") String authHeader) throws Exception {
    DecodedJWT jwt = this.checkJWT(authHeader);
    List<String> roles = ((List)jwt.getClaim( name: "realm_access").asMap().get("roles"));

    if(roles.contains("student")){
        String username = jwt.getClaim( name: "preferred_username").asString();
        String json = Files.lines(Paths.get( first: "src/main/resources/esamiByStudenti.json"))
                .collect(Collectors.joining());
        JSONObject jsonObject = new JSONObject(json);
        JSONArray studentsData = jsonObject.getJSONObject(username).getJSONArray( name: "esami_prenotati");
        ObjectMapper mapper = new ObjectMapper();
        List<Exam> exams = new ArrayList<>();

        if (studentsData != null) {
            for (int i=0;i<studentsData.length();i++){
                Exam e = mapper.readValue(studentsData.get(i).toString(), Exam.class);
                exams.add(e);
            }
        }
        return exams.stream().map(Exam::toString).collect(Collectors.toList());
    } else{
        throw new BusinessException(ResponseCode.FORBIDDEN);
```

Figura 3.24: Implementazione del metodo `getExamsRegistration`.

L'elenco di esami trovato viene inserito una lista e ad ognuno si applicherà il metodo `toString()`, in modo da ottenere il seguente output:

```
[
    "Esame {codice='1', data='02/02/2022', materia='Tecnologie
        ↪ Web'}",
    "Esame {codice='2', data='10/07/2022', materia='Tecnologie Web'}"
]
```

Listato 3.6: Body della response della request `GetExamsRegistration`.

GetAllExams

Il metodo `GetAllExams` effettua gli stessi controlli dell'altro ma, al contrario del precedente, fornisce accesso alla risorsa se l'utente che ne ha fatto richiesta è un `teacher`. La risorsa, in questo caso è la lista di tutti gli esami.

```
@GetMapping(path="/mostraEsami", produces = MediaType.APPLICATION_JSON_VALUE)
public List<String> getAllExams(@RequestHeader("Authorization") String authHeader) throws Exception
    DecodedJWT jwt = this.checkJWT(authHeader);
    List<String> roles = ((List) jwt.getClaim( name: "realm_access").asMap().get("roles"));
    if (roles.contains("teacher")) {
        String json = Files.lines(Paths.get( first: "src/main/resources/esami.json"))
                .collect(Collectors.joining());
        JSONObject jsonObject = new JSONObject(json);
        JSONArray allExams = jsonObject.getJSONArray( name: "esami");
        ObjectMapper mapper = new ObjectMapper();
        List<Exam> exams = new ArrayList<>();

        if (allExams != null) {
            for (int i = 0; i < allExams.length(); i++) {
                Exam e = mapper.readValue(allExams.get(i).toString(), Exam.class);
                exams.add(e);
            }
        }
        return exams.stream().map(Exam::toString).collect(Collectors.toList());
    } else {
        throw new BusinessException(ResponseCode.FORBIDDEN);
```

Figura 3.25: Implementazione del metodo `getAllExams`.

E' possibile notare che i controlli sul ruolo vengono effettuati alla stessa maniera di prima, ciò che cambia è la struttura della risorsa.

Il file da cui attingere per reperire tutte le informazioni sugli esami esistenti, infatti, è `esami.json`. Se la request va a buon fine si ottiene la seguente response:

```
[
    "Esame {codice='1', data='02/02/2022', materia='Tecnologie
        ↪ Web'}",
    "Esame {codice='2', data='10/07/2022', materia='Tecnologie
        ↪ Web'}",
    "Esame {codice='3', data='11/02/2022', materia='Reti'}"
]
```

Listato 3.7: Body della response della request `GetAllExams`.

3.8 Gestione delle eccezioni

Per la eccezioni è stata sfruttata, in entrambe le applicazioni, la classe `ResponseEntityExceptionHandler` che centralizza la gestione delle eccezioni derivate dai metodi annotati con il tag `@ExceptionHandler`.

Inoltre, per customizzare le response è stata definita una classe `BusinessException` che associa uno specifico `ResponseCode` ad una `Exception`. In questo modo, quando si verifica un problema si ha in risposta un messaggio chiaro ed esaustivo a riguardo, anzichè dei semplici codici `HTTP` (200 quando la richiesta va a buon fune, 400 altrimenti).

Conclusioni

Il problema del riconoscimento dell'utente e dell'attribuzione dei permessi a lui associati è un problema che si verifica laddove vengono fornite particolari risorse informatiche (risorse web, accesso a Internet, posta elettronica, ecc.) che richiedono fasi di autenticazione e di autorizzazione.

Tra gli obiettivi del progetto di tesi, vi era proprio la realizzazione di un sistema che riconoscesse gli utenti e differenziasse gli accessi in base ad alcune informazioni, come ad esempio il ruolo. Keycloak è uno strumento che si è rivelato molto utile in questo ambito.

Tutti gli obiettivi iniziali sono stati raggiunti. In primo luogo, infatti, è stato fatto uno studio dell'arte rispetto ai sistemi di Identity and Access management e a tutto ciò che concerne l'identità digitale (che cos'è, perchè è importante, il suo ciclo di vita).

In secondo luogo, è stato installato un Identity Manager Keycloak per l'autenticazione e la gestione delle utenze.

In aggiunta, poichè Keycloak può essere configurato per delegare l'autenticazione a uno o più Identity Provider, è stato simulato il social login con GitHub.

Infine, coerentemente con gli obiettivi iniziali, per quanto riguarda la configurazione di un SSO tra due applicazioni, sono state dapprima realizzate due API diverse che si connettessero alle istanze di Keycloak (una per ciascuna applicazione) e, in seguito, sono state inviate delle request sfruttando gli access token. In questo modo, la seconda applicazione è in grado di riconoscere tutti gli utenti e i rispettivi ruoli derivati dalla prima applicazione e di consentire loro o di negare l'accesso ad altre risorse, senza che questi abbiamo fatto una seconda autenticazione.

In ottica futura, si potrebbe pensare di integrare questo progetto con il protocollo LDAP. Keycloak, infatti, ha un supporto built-in per connettersi a dei server LDAP già esistenti. LDAP consente di archiviare, gestire e proteggere informazioni quali password e nomi utente dell'organizzazione e dei rispettivi utenti e risorse. Questo protocollo semplifica l'accesso all'archivio perché organizza le informazioni in modo gerarchico ed è fondamentale per le aziende in crescita che acquisiscono e devono gestire sempre più dati e risorse.

Ringraziamenti

3.9 Considerazioni finali sull'esperienza

Quando mi è stato proposto da Imola Informatica di scegliere tra una serie di progetti diversi, ho individuato proprio questo perchè ho trovato l'argomento particolarmente interessante. Il social login e il single sign-on sono degli strumenti con cui si ha spesso a che fare nella vita quitidiana, quindi è stato utile ed entusiasmante comprendere a fondo la loro importanza e il loro funzionamento nel dettaglio. Mi è stato delineato abbastanza bene il contesto sul quale lavorare e mi sono state fornite delle indicazioni ben precise per quanto riguarda la scelta degli strumenti da utilizzare. Il tutto, lasciandomi ampio e completo spazio sulla realizzazione del progetto e sulla gestione delle ore a disposizione.

Nell'ottica di proporre non solo un buon progetto, ma anche di imparare qualcosa di nuovo e di ampliare le mie conoscenze, ho cercato di approfondire al meglio certi concetti, sia come cruccio personale ma anche, talvolta, obbligato dal cambio di versione delle tecnologie impiegate.

Sono convinta del fatto che se avessi svolto il progetto in azienda, stando a contatto con tutor e altri colleghi, avrei impiegato meno tempo a risolvere eventuali dubbi e problematiche. Con questa considerazione non intendo attribuire la colpa a nessuno in particolare, considerato che è stato stabilito di comune accordo di procedere in modalità smart-working.

Mi ritengo molto soddisfatta del lavoro svolto, malgrado le ovvie difficoltà dovute ad un'esperienza del tutto nuova e ad un argomento di tesi di cui conoscevo ben poco. Il mio augurio è di poter sfruttare le conoscenze e le competenze che ho acquisito durante tutto il mio percorso universitario, con particolare riferimento a quelle capacità che possono solo scaturire dall'esperienza lavorativa assaporata durante il tirocinio.

3.10 Ringraziamenti

Affrontare il secondo anno di università, quello da tutti considerato il più impegnativo, durante la pandemia non è stato facile. Il carico di studio è stato raramente alleggerito, nonostante le criticità riscontrate nelle lezioni online. I laboratori online sono stati penalizzanti e difficili da seguire, anche se i professori si sono resi sempre molto disponibili per eventuali chiarimenti. Con il passare delle settimane, però, la mancanza della routine a cui ogni studente era abituato e l'assenza del contatto con compagni e professori ha iniziato a pesare sempre di più. Ritengo che, a differenza degli studenti di altri gradi di istruzione, noi universitari siamo stati lasciati non uno, ma cento passi indietro.

Dedico questa tesi soprattutto a me stessa, per non essermi mai arresa e per essere giunta al termine di questo percorso, sebbene ci abbia impiegato più tempo del dovuto.

Un ringraziamento speciale va alla mia famiglia che mi ha sempre sostenuto, senza mettermi fretta e a Nicola, che è rimasto al mio fianco quando nessun altro ne sarebbe stato in grado.

Bibliografia

[1] Juanita Blue, Joan Condell, and Tom Lunney. A review of identity, identification and authentication. *Int. J. Inf. Secur. Res*, 8:794–804, 2018.

[2] Thomas Bröker. How to support an effective digital identity lifecycle, 2022. [Image].

[3] Digital identity guidelines. `https://pages.nist.gov/800-63-3/`. Accessed: 2023-02-02.

[4] Uber team. Security update. Technical report, Uber, `https://www.uber.com/newsroom/security-update/`, September 2022. available online.

[5] Digital Guide IONOS team. Autenticazione kerberos: spiegazione.

[6] Daniel Sonck. Kerberos negotiations, 2011. [Image].

[7] Bruno Marafini. Single sign-on, accesso facilitato alle risorse di rete: ecco come funziona, 2019. [Image 1].

[8] Bruno Marafini. Single sign-on, accesso facilitato alle risorse di rete: ecco come funziona, 2019. [Image 2].

[9] Verizon. 2022 data breach investigations report. Technical report, 2022. `https://www.verizon.com/business/resources/reports/dbir/`.

[10] Bruno Marafini. Single sign-on, accesso facilitato alle risorse di rete: ecco come funziona, 2019. [Image 3].

[11] Bruno Marafini. Single sign-on, accesso facilitato alle risorse di rete: ecco come funziona, 2019. [Image 4].

[12] David Kirkpatrick. Social media marketing: Social login or traditional website registration?

[13] D. Hardt. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. `http://www.rfc-editor.org/rfc/rfc6749.txt`.

[14] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.

[15] Introduction to json web tokens. https://jwt.io/introduction. Accessed: 2023-01-31.

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, RFC Editor, May 2008. http://www.rfc-editor.org/rfc/rfc5280.txt.

[17] Json web token (jwt). https://www.iana.org/assignments/jwt/jwt.xhtml. Accessed: 2023-02-01.

[18] M. Jones, J. Bradley, and N. Sakimura. Json web signature (jws). RFC 7515, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7515.txt.

[19] VMware Inc.