Alma Mater Studiorum - Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

SINGLE SIGN-ON PER L'AMBIENTE CAS

Relatore:
Chiar.mo Prof.
PAOLO CIANCARINI

Presentata da:
FRANCESCA GUERZONI

Correlatore:
Prof.
MARCELLO MISSIROLI

Sessione IIa - secondo appello
Anno Accademico 2020-21

Introduzione

L'argomento centrale trattato nel lavoro di tesi seguente è il Single Sign-On, in particolare la teoria su cui si basa ed il suo inserimento in un sistema esistente.
Innanzitutto, verrà introdotto il tema dell'identità in generale e la sua applicazione al contesto del software. Come prima cosa vengono quindi date diverse definizioni di termini che verranno utilizzati nelle parti successive. Dopo di che andremo a mostrare e spiegare nel dettaglio il ciclo di vita di un'identità in un sistema informatico; in questo modo vengono anche chiariti diversi concetti importanti. Poi approfondiremo alcuni protocolli di gestione delle identità che possono risultare utili ad uno sviluppatore nelle fasi di autenticazione ed autorizzazione in un'applicazione. Vedremo anche un approfondimento sul ruolo del Single Sign-On, il suo funzionamento generale ed i vantaggi e svantaggi del suo utilizzo.
Successivamente vedremo il software Keycloak, lo strumento di gestione di identità ed accesso selezionato per essere successivamente utilizzato per fornire il servizio di Single Sign-On. Prima verranno presentati diversi concetti importanti, poi verranno elencate e spiegate le funzionalità principali che questo software fornisce.
Faremo riferimento all'ambiente di sviluppo CAS, in particolare la sua versione 2.0, ospitato su un server della rete dipartimentale DISI. Spiegheremo la sua architettura ed i servizi open source che mette a disposizione, utili in particolare per chi lavora seguendo un modello di sviluppo software agile. CAS verrà usato come sistema esistente sul quale svolgere il lavoro, ovvero aggiungere un Single Sign-On. A seguire spiegheremo le fasi attraverso le quali Keycloak è stato inserito in CAS ed il modo in cui alcuni servizi di CAS, ovvero Jenkins, SonarQube e GitLab sono stati configurati per abilitare ed usare Keycloak come SSO. Verrà anche mostrato come la struttura di CAS è cambiata al seguito delle operazioni sovracitate.
Infine, concluderemo spiegando quanto possa risultare importante l'utilizzo di un Single Sign-On, presentando anche le problematiche riscontrate durante il lavoro effettuato ed i possibili ulteriori miglioramenti che potrebbero essere applicati utilizzando Keycloak su CAS.

Indice

Elenco delle figure

Listings

CAPITOLO 1

Identità e Single Sign-On

1.1 Identità ed il suo ciclo di vita

Il contenuto di questa sezione si ispira al libro [15].
Per capire il processo di gestione degli utenti autorizzati a usare un'applicazione, la nozione di identità è fondamentale, come anche il suo ciclo di vita.
Innanzitutto, chiariamo la terminologia utilizzata.

Identificatore

Si può utilizzare un identificatore per identificare entità umane e non umane (ad esempio bot o dispositivi). Consiste di un singolo attributo e può essere un indirizzo email, il numero della patente di guida o in generale un codice che contraddistingue in modo univoco la persona. Per le entità non umane è solitamente utilizzata come identificatore una stringa alfanumerica di caratteri che viene associata al momento della creazione/registrazione. Nella gestione dell'identità risulta pertanto fondamentale il ruolo dell'identificatore.

Identità

Per parlare di identità di una persona o entità in uno specifico contesto è necessario definire un insieme di attributi, insieme che contiene uno o più identificatori oltre ad altri attributi che si riferiscono alla persona/entità, come ad esempio il nome, l'età o l'indirizzo. Per potersi autenticare o per autorizzare le applicazioni all'uso e alla trasmissione di informazioni si utilizzano gli attributi che costituiscono l'identità. Le identità online si riferiscono a contesti quali applicazioni o suite di applicazioni; devono contenere almeno un identitificatore e alcuni attributi associati alla persona/entità definita come utente. Altri tipi di identità sono l'identità lavorativa o l'identità personale, che conter-

ranno ovviamente attributi differenti a seconda dell'ambiente e del contesto a cui fanno riferimento.

Account

Per poter eseguire delle operazioni all'interno di una applicazione, o una suite di applicazioni, bisogna definire un account, che è un costrutto locale associato ad una specifica identità. Gli attributi di questa identità verranno salvati internamente all'applicazione oppure saranno archiviati in oggetti separati ed utilizzati tramite un riferimento. L'identificatore dell'account può essere diverso da quello dell'identità associata.
Quando si effettua il login ad una applicazione, quindi, viene usato un account a cui è associata una identità con i suoi attributi, e se il login termina con successo è possibile eseguire azioni all'interno del sistema.

Identity Management System

L'insieme dei servizi che sovrintendono alla creazione, modifica e rimozione di account e di identità è l'Identity Management System (IdM). Viene utilizzato anche per l'autenticazione e per autorizzare l'accesso alle risorse, quindi per proteggere le risorse da accessi non autorizzati.

1.1.1 Ciclo di vita di un'identità

Di seguito parleremo degli eventi considerati come i principali nel ciclo di vita di una identità, entrando nel dettaglio delle varie fasi.

Figura 1.1: Ciclo di vita di un'identità[15]

In questa sezione verranno spiegati tutti gli eventi mostrati in Fig.1.1, ad eccezione del Single Sign-On che avrà una parte dedicata, la sezione 1.3.

1.1.1.1 Provisioning

Per accedere ad un'applicazione e alle sue risorse protette gli utenti devono autenticarsi e ottenere l'autorizzazione all'accesso. La fase denominata *provisioning* ha come obiettivo

la creazione o la selezione di un insieme di account, contenente le informazioni di identità utilizzate per l'autenticazione.
Durante il provisioning vengono acquisiti utenti e creati account e profili di identità. Gli approcci che si possono adottare per svolgere questo compito sono:

- Creare una nuova identità attraverso la compilazione di un modulo da parte dell'utente.
- Mandare un invito a uno specifico gruppo di utenti.
- Trasferire le identità degli utenti da un insieme già esistente.
- Utilizzare l'identità di un utente presente in un identity provider.
- La creazione diretta delle identità da parte di un amministratore o un processo automatizzato.

A volte risulta conveniente usare una combinazione degli approcci sopra elencati.

1.1.1.2 Autorizzazione

Nel momento in cui un account viene creato, se ne definiscono anche i privilegi, che regolano ciò che l'account è o non è autorizzato a fare. La concessione dei privilegi viene definita *autorizzazione*, e può essere aggiornata nel tempo. Il processo di controllo degli accessi, infatti, viene suddiviso in due fasi distinte:

- la definizione delle *policy*, che contiene la funzione che specifica i privilegi di accesso alle risorse, ovvero la funzione di autorizzazione;
- la fase della loro applicazione, dove si approvano o rifiutano le richieste di accesso proprio in base alle autorizzazioni definite in precedenza.

Le policy di autorizzazione possono basarsi sugli attributi del profilo utente o su fattori dinamici valutati nel momento in cui un utente si autentica o effettua una richiesta.

1.1.1.3 Autenticazione

L'autenticazione è la procedura con la quale un utente può accedere a tutte quelle risorse e quelle funzionalità che sono private, cioè non disponibili pubblicamente. Per effettuare l'autenticazione è necessario prima di tutto scegliere con quale account si vuole accedere, ovviamente questo account deve essere già stato registrato nella fase di previsioning. Per effettuare l'autenticazione vengno utilizzate delle credenziali di accesso che comprendono un identitificatore dell'account ed una caratteristica strettamente personale dell'utente, che consiste ad esempio in una password (ciò che l'utente sa), un codice (generato da

un dispositivo) o una informazione biometrica (come, ad esempio, l'impronta digitale). Possono anche essere richieste combinazioni di più di una di queste informazioni. Tali credenziali saranno accettate solamente se il sistema le riconosce come corrispondenti a quelle già presenti in esso.

1.1.1.4 Applicazione delle policy di accesso

Qualunque azione venga intrapresa da un utente, deve risultare dal controllo dei privilegi concessi all'account che essa sia permessa. Questa è la fase di applicazione delle policy di accesso, in cui ogni richiesta di accesso viene valutata e quindi approvata o rifiutata. Questo significa che l'autorizzazione deve definire cosa può fare un utente/entità, mentre l'applicazione delle policy di accesso verifica, attraverso il controllo dei privilegi assegnati, che le azioni richieste dall'utente siano permesse.

1.1.1.5 Sessioni

Dopo essersi autenticato ed avere ricevuto le autorizzazioni dal sistema, l'utente è abilitato ad eseguire specifiche azioni all'interno dell'applicazione ma solo per un periodo di tempo limitato. Questo tempo viene determinato principalmente dalla qualità e dalla sensibilità dei dati che sono gestiti dall'applicazione (pensiamo ad una applicazione bancaria, ad esempio). Terminato questo periodo di tempo, all'utente viene chiesto di autenticarsi di nuovo. La sessione conserva, quindi, non solo un identificatore dell'utente ma anche altre informazioni quali l'avvenuta autenticazione, il meccanismo di autenticazione, il livello di sicurezza, il momento in cui è avvenuta l'autenticazione. Questo permette all'applicazione di conoscere il momento preciso in cui richiedere una nuova autenticazione. L'intervallo di tempo durante il quale l'utente è attivo e può eseguire azioni all'interno dell'applicazione viene definito limite o timeout di sessione, ed è tanto più breve quanto maggiore sono sensibili i dati elaborati.

1.1.1.6 Rafforzamento dell'autenticazione

La classica forma di autenticazione, tramite nome utente e password, viene oggi considerata ormai debole, in quanto viene coinvolto un solo fattore (la password) che risulta relativamente semplice da ottenere o ricavare per un uso non autorizzato dell'account. Se viene usata un'autenticazione che coinvolge più fattori (MFA - *multi-factor authentication*) viene utilizzato, oltre alla password, qualcosa che l'utente ha (per esempio un altro dispositivo, come il telefono) e/o qualcosa che l'utente è (per esempio un fattore biometrico, come l'impronta digitale).
Un caso differente, invece, è quando un utente che ha effettuato il login con nome utente e password, ha in seguito la necessità di gestire informazioni con un livello di sensibilità e requisiti di autenticazione più elevati. Questo scenario, chiamato *autenticazione step-up*, richiede l'atto di autenticare l'utente con una forma più forte, elevando così una

sessione esistente ad una di livello di sicurezza maggiore. In questo caso potrebbero essere richieste ulteriori credenziali, come ad esempio una password monouso.

1.1.1.7 Logout

Il logout si differenzia dal timeout di una sessione per il fatto che in questo caso è l'utente stesso che chiede di terminare la sessione, mentre nel caso in cui avvenga un timeout di sessione l'applicazione ha la facoltà di scegliere se mantenere la sessione in stato di sospensione, così da ricostruirla nel caso in cui l'utente chieda una nuova autenticazione. Di norma, quando un utente termina di utilizzare l'applicazione dovrebbe effettuare il logout, in modo da doversi nuovamente autenticare per ottenere l'accesso nel caso in cui volesse tornare ad usare l'applicazione. Se viene usato il Single Sign-On potrebbero esserci più sessioni da terminare e stabilire quali sessioni terminare in caso di disconnessione dell'utente è una decisione di tipo progettuale.

1.1.1.8 Gestione e recupero dell'account

Gli utenti e gli amministratori possono visualizzare ed aggiornare i vari attributi di un utente attraverso quell'insieme di funzionalità e processi che si identificano nella gestione degli account. Infatti, durante l'esistenza di un'identità può essere necessario modificare degli attributi del profilo come, ad esempio, un indirizzo o un numero di telefono. Non solo i dati personali sono soggetti a cambiamenti, all'interno di una azienda un utente potrebbe cambiare ruolo e quindi il suo profilo dovrebbe subire modifiche per rappresentare la nuova posizione o privilegio.
Nel caso in cui l'utente dimentichi la password o perda il dispositivo che serve al processo di autenticazione si rende necessario definire nuove credenziali. Il meccanismo che permette all'utente di confermare la legittima proprietà dell'account si chiama "Recupero dell'account". In questo caso è indispensabile utilizzare un mezzo alternativo per poter confermare la proprietà dell'account prima di dare il permesso alla creazione di nuove credenziali. Anche un identity provider può essere delegato a controllare questa fase del ciclo di vita di un identità.

1.1.1.9 Deprovisioning

Quando un account deve essere chiuso si apre la fase di *deprovisioning*. Ci sono diversi motivi per cui un account deve essere chiuso:

- il proprietario non ha più bisogno del servizio, quindi ne chiede la cancellazione;
- il cliente viola i termini del servizio o ne abusa, determinando la richiesta di chiusura da parte dell'amministratore stesso del servizio;

- cessano le condizioni che sono un prerequisito per l'utilizzo del servizio (un dipendente lascia il lavoro, uno studente termina il ciclo di studi, ...)

Qualunque sia il motivo che porta alla disattivazione dell'account, non deve essere più possibile utilizzarlo per accedere alle risorse del servizio. Chiudere un account significa che si devono disabilitare tutte le informazioni sull'dentità ad esso associata in modo tale da non poterlo più usare. Il deprovisioning può eliminare l'account e tutte le informazioni di identità associate oppure solo disabilitarle, così da rendere possibile preservare i dati e poterli riutilizzare in seguito.

1.2 Principali protocolli

Anche per questa sezione prendo a modello il libro [15].

1.2.1 OAuth 2.0

OAuth 2.0 è un framework di autorizzazione pubblicato nel 2012 che consente ad un'applicazione di utilizzare le API di terze parti. L'uso di questo framework permette all'applicazione stessa di ottenere l'autorizzazione all'uso di API esterne nel caso in cui possieda il contenuto a cui vuole accedere, ma evita anche all'utente di fornire le proprie credenziali.
OAuth 2.0 venne infatti progettato ed introdotto principalmente per questo secondo caso, dando anche la possibilità all'utente di limitare le azioni che l'applicazione può svolgere. Un utente ha spesso la necessità di consentire ad una applicazione di accedere ad una sua risorsa protetta, appartenente ad un sito separato. Prima della creazione di questo framework, con l'obbligo di fornire le proprie credenziali di accesso all'applicazione, per essa era possibile ottenere informazioni, modificare o cancellare dati di qualunque tipo. L'applicazione doveva inoltre conservare le informazioni per accessi successivi, aumentando quindi il rischio di accessi non autorizzati in caso di compromissione dell'applicazione. Inoltre, l'unico modo per revocare l'utilizzo di tali credenziali era la loro modifica, compromettendo però in questo modo anche l'accesso di eventuali altre applicazioni.

1.2.1.1 Funzionamento

Il funzionamento di OAuth 2.0 è mostrato in Fig.1.2.

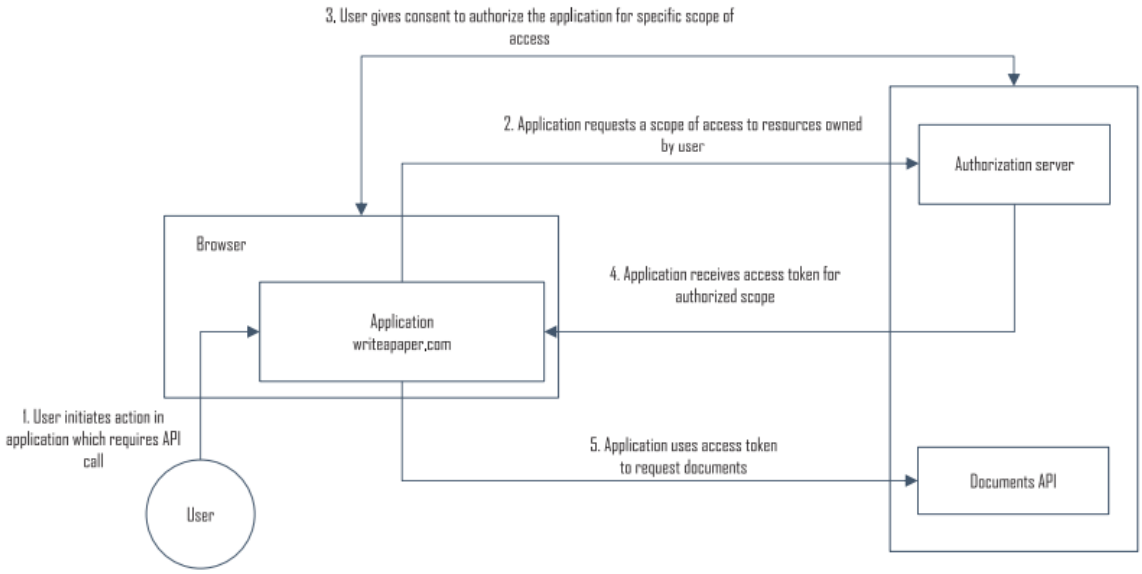

Figura 1.2: Funzionamento di OAuth 2.0[15]

1. All'interno dell'applicazione l'utente effettua un'operazione che necessita di chiamare delle API esterne.

2. L'applicazione invia una richiesta di autorizzazione per le API ad un apposito server, il server di autorizzazione. La richiesta deve indicare cosa vuole richiedere alle API, lo scope.

3. Se il contenuto a cui l'applicazione vuole accedere appartiene all'utente, il server di autorizzazione autentica l'utente e gli chiede di consentire all'applicazione l'accesso ai dati richiesti.

4. Se l'utente dà il consenso, l'applicazione riceve dal server un access token, che le permette di fare richieste alle API all'interno dello scope indicato in precedenza.

5. L'applicazione usa l'access token per accedere ai dati che le servono.

Bisogna ricordare che il protocollo OAuth 2.0 fornisce una soluzione per l'autorizzazione, ma non per l'autenticazione; infatti l'access token non è usato per trasmettere informazioni sull'evento di autenticazione o sull'utente. Per autenticare un utente può essere usato OIDC, descritto nella prossima sezione.

1.2.1.2 Ruoli

OAuth 2.0 definisce quattro ruoli coinvolti in una richiesta di autorizzazione:

- **Resource Server**: espone delle API e mantiene protette le risorse a cui l'applicazione deve accedere.
- **Resource Owner**: utente o entità che possiede le risorse mantenute dal *resource server*.
- **Client**: applicazione che deve accedere alle risorse del *resource server*, per conto proprio o per conto del proprietario della risorsa. Esistono due diversi tipi di client:
  - **Confidential**: applicazione che lavora su un server protetto e può archiviare in modo sicuro infomazioni riservate per autenticarsi ad un server di autorizzazione, oppure utilizzare altri meccanismi di autenticazione, purché in modo sicuro (ad esempio un'applicazione web in cui il codice viene eseguito su un server back-end protetto).
  - **Public**: applicazione che viene eseguita principalmente sul dispositivo dell'utente (applicazione nativa) o sul browser e non può archiviare in modo sicuro informazioni riservate o utilizzare altri mezzi per autenticarsi.
- **Authorization Server**: ha il compito di autorizzare le applicazioni a chiamare le API del *resource server*, e deve quindi essere considerato affidabile da quest'ultimo. L'Authorization server autentica il proprietario delle risorse e gli chiede il consenso a concederne l'accesso.

1.2.1.3 Richiedere l'autorizzazione

OAuth 2.0 definisce quattro metodi con cui le applicazioni ottengono l'autorizzazione ad utilizzare un'API. Ognuno di essi utilizza un diverso tipo di credenziali ed il tipo da utilizzare dipende dal caso d'uso e dal tipo di applicazione. Questi metodi sono noti come "authorization grants".
I quattro tipi di authorization grants sono i seguenti:

- **Authorization Code Grant**: questo tipo di authorization grant consiste in due diverse richieste da parte del client. Il flusso seguito per richiedere l'autorizzazione è il seguente:
  1. L'utente accede all'applicazione.
  2. L'applicazione reindirizza il browser all'authorization server con una richiesta di autorizzazione.

3. L'authorization server richiede all'utente di autenticarsi e contestualmente di approvare la richiesta dell'applicazione.
4. L'utente si autentica e autorizza l'accesso alle informazioni.
5. L'authorization server reindirizza il browser all'URL di callback dell'applicazione, che era già stato definito in precedenza, con l'authorization code.
6. L'applicazione chiama l'endpoint dell'authorization server che serve per ottenere i token e gli passa il codice di autorizzazione.
7. L'authorization server risponde con un access token ed eventualmente un refresh token, che può essere utilizzato per richiedere un nuovo access token quando quello vecchio scade.
8. L'applicazione chiama le API del resource server, usando l'access token.

Normalmente, l'Authorization Code Grant dovrebbe essere usato solo da un confidential client, altrimenti l'authorization code potrebbe essere intercettato da una applicazione diversa, che potrebbe in questo modo ottenere un accesso non autorizzato a risorse private. Per evitare questa evenienza, infatti, l'applicazione è obbligata ad autenticarsi all'authorization server. Quando si è reso necessario usare l'Authorization Code Grant anche con un public client, allora è stato aggiunto il PKCE (Proof Key for Code Exchange), che viene utilizzato per avere la certezza che l'applicazione che riceve il codice sia la stessa che lo userà in seguito per ottenere un access token. Di seguito la descrizione per punti del comportamento di PKCE:

- l'applicazione crea una stringa random, chiamata *code verifier*, sufficientemente lunga da fornire protezione contro gli attacchi che mirano ad indovinarla;

- da tale stringa l'applicazione ricava un valore, il *code challenge*, che viene inviata all'authorization server nella prima richiesta, insieme al metodo utilizzato per la trasformazione;

- nella seconda richiesta che contiene l'authorization code, l'applicazione aggiunge il code verifier.

L'Authorization Server, dopo quest'ultima richiesta, ricalcola il code challenge usando il code verifier e il metodo di trasformazione comunicato nella prima richiesta, controllando che le informazioni siano coerenti con i dati già in suo possesso. Solo in questo modo c'è la sicurezza che il client, il solo a conoscere il code verifier, che riceve l'access token sia quello che lo ha richiesto.

- **Implicit Grant**: il suo funzionamento è lo stesso dell'Authorization Code Grant ma senza il coinvolgimento dell'Authorization Code, quindi non si esegue la doppia richiesta. Adesso non è più consigliato il suo utilizzo, veniva principalmente usato

quando nei browser era poco supportato lo standard CORS (Cross-Origin Resource Sharing).

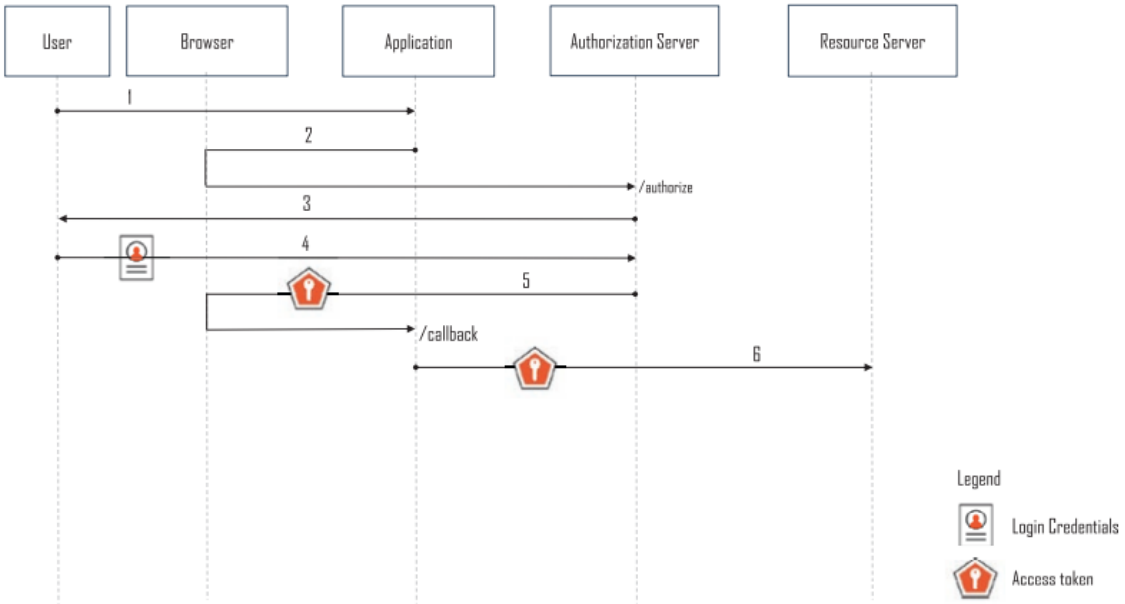

Figura 1.3: Funzionamento dell'Implicit Grant di OAuth 2.0[15]

1. L'utente accede all'applicazione.
2. L'applicazione reindirizza il browser all'endpoint di autorizzazione dell'authorization server con una richiesta di autorizzazione.
3. L'authorization server richiede all'utente di autenticarsi e di dare il suo consenso alla richiesta dell'applicazione.
4. L'utente si autentica e dà il suo consenso.
5. L'authorization server risponde con un access token.
6. L'applicazione chiama le API del resource server, usando l'access token.

- **Resource Owner Password Credentials Grant**: viene utilizzato in situazioni in cui l'applicazione risulta affidabile nella gestione delle credenziali dell'utente e non è possibile usare altri tipi di authorization grant. Il funzionamento è il seguente:

  - il client raccoglie le credenziali dell'utente direttamente senza reindirizzarlo all'atuhorization server e quindi alla pagina di login;

  - l'applicazione passa le credenziali all'authorization server come parte della sua richiesta che mira ad ottenere l'access token.

Il Resource Owner Password Credentials Grant comunica le credenziali dell'utente all'applicazione, generando quindi i problemi che sono stati già descritti in precedenza, e di conseguenza se ne sconsiglia l'uso a meno che non si debba effettuare una migrazione di utenti. In generale, quando si usa questo tipo di grant è consigliata la cancellazione delle credenziali non appena l'applicazione ottiene l'access token, così da rendere minimo il rischio che esse vengano compromesse.

- **Client Credentials Grant**: nel caso in cui un'applicazione debba accedere a risorse che sono già in suo possesso, non sono necessarie interazioni tra l'utente e l'authorization server. Con l'uso di questo tipo di grant l'applicazione usa le proprie credenziali per autenticarsi e ottenere un access token, utilizzando il Client ID e il Client Secret che ha ottenuto nel momento in cui si è registrata all'Authorization Server.

1.2.2 OpenID Connect

OpenID Connect (OIDC) è un protocollo di autenticazione basato su OAuth 2.0 (protocollo che gestisce l'autorizzazione ma non l'autenticazione degli utenti). Viene progettato per consentire agli authorization server di autenticare gli utenti per conto delle applicazioni e restituirne i risultati. OIDC consente cioè ad un'applicazione di delegare l'autenticazione di un utente ad un authorization server OAuth 2.0, il quale le resituisce informazioni relative all'utente e all'evento di autenticazione in un formato standard.

1.2.2.1 Ruoli

Esistono diversi ruoli coinvolti nell'utilizzo di OIDC:

- **End User**: l'utente che deve essere autenticato.
- **OpenID Provider (OP)**: authorization server OAuth 2.0 che implementa OIDC (può autenticare un utente e restituire delle informazioni su esso e sull'evento di autenticazione ad un'applicazione).
- **Relying Party (RP)**: client OAuth 2.0 che delega l'autenticazione di un utente ad un OpenID provider e gli richiede delle informazioni su di esso.

1.2.2.2 Endpoints

OIDC utilizza l'endpoint di autorizzazione e quello per ottenere i token, descritti nella sezione su OAuth 2.0, e ad essi aggiunge l'endpoint delle informazioni dell'utente, chiamato UserInfo endpoint. Quest'ultimo restituisce dati su un utente autenticato. La chiamata a questo endpoint richiede un access token, che definisce anche quali dati verranno restituiti.

1.2.2.3 Funzionamento

La Fig.1.4 mostra i passi eseguiti da OIDC per autenticare un utente.

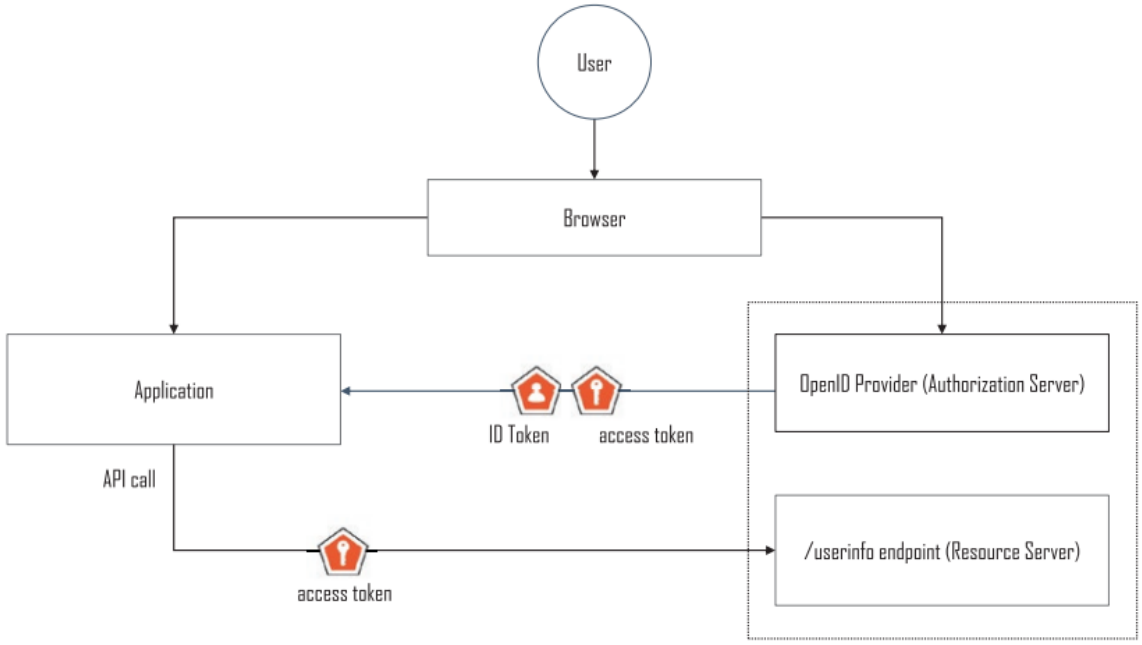

Figura 1.4: Funzionamento di OpenID Connect[15]

1. L'utente accede all'applicazione (Relying Party).
2. L'applicazione reindirizza il browser ad un authorization server che implementa OIDC, un OpenID Provider.
3. L'OpenID Provider interagisce con l'utente per autenticarlo, supponendo che non abbia ancora effettuato l'accesso.
4. Dopo l'autenticazione, il browser viene reinderizzato all'applicazione con i security token richiesti.
5. L'applicazione può utilizzare uno dei seguenti metodi per ricevere le informazioni di autenticazione di un utente:
   - **Richiedere un ID token**, cioè un token utilizzato per trasmettere dati su un evento di autenticazione ed un utente autenticato ad una relying party.
   - **Richiedere un access token**, che verrà inviato all'endpoint UserInfo, dal quale otterrà le informazioni.

Un'applicazione può utilizzare un provider OpenID sia per l'autenticazione che per l'autorizzazione degli utenti perché OIDC è un livello superiore di OAuth 2.0.

1.2.2.4 Flussi OIDC

I flussi OIDC, che hanno somiglianze con i grant type definiti in OAuth 2.0, sono progettati rispetto ai vincoli di diversi tipi di applicazioni. I flussi principali sono i seguenti:

- **Authorization Code Flow**: simile all'Authorization Code Grant di OAuth 2.0, si basa su due richieste e un authorization code. Di seguito, i passi svolti durante questo flusso:

  1. L'utente accede all'applicazione (relying party).
  2. Il browser dell'utente viene reindirizzato all'OpenID provider con una richiesta di autenticazione.
  3. L'OpenID provider interagisce con l'utente per autenticarlo e per ottenere il consenso per lo scope della richiesta di informazioni dell'utente. Tale scope specifica quali sono le informazioni richieste sull'utente autenticato.
  4. L'utente si autentica, dà il consenso e l'OpenID provider crea o aggiorna una sessione di autenticazione per l'utente.
  5. Il browser dell'utente viene reindirizzato all'applicazione con l'authorization code.
  6. L'applicazione invia una richiesta al token endpoint dell'OpenID provider con l'authorization code.
  7. L'OpenID provider risponde con un access token, un ID token ed eventualmente un refresh token.
  8. L'applicazione può utilizzare l'access token, inserendolo nella richiesta all'endpoint UserInfo dell'OpenID provider.

  Per poter eseguire la seconda chiamata all'endpoint e ottenere così i token è necessario che l'applicazione sia in grado di autenticarsi presso l'OpenID provider. I public client, che non hanno un livello di sicurezza sufficiente per i dati sensibili per poter effettuare tale autenticazione, possono utilizzare PKCE come già descritto nella sezione su OAuth 2.0 e come mostrato nella seguente immagine.

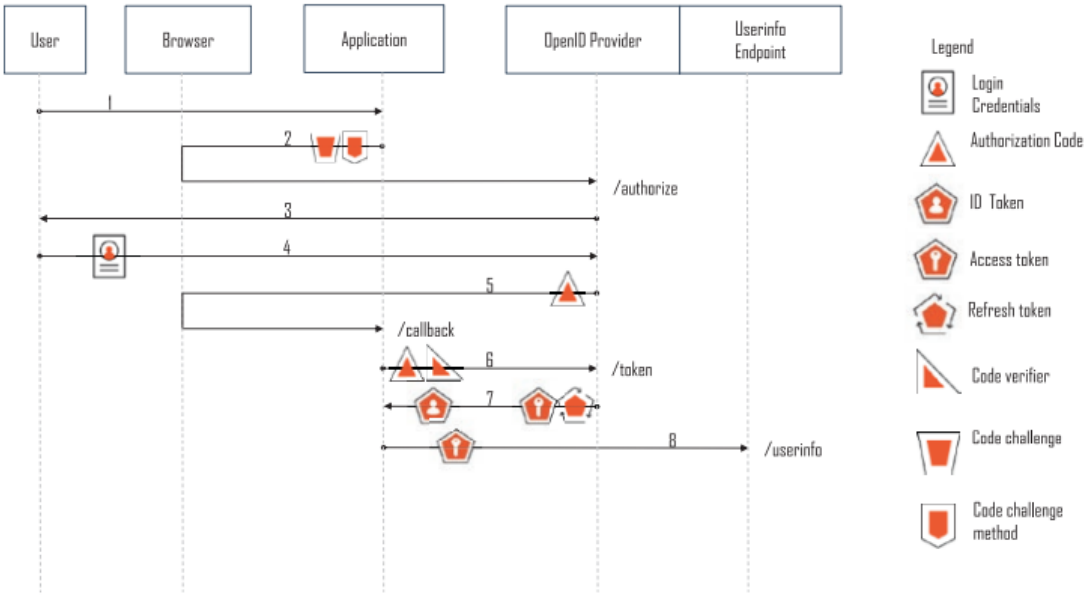

Figura 1.5: Funzionamento dell'Authorization Code Flow di OpenID Connect[15]

Come si vede il funzionamento è lo stesso descritto per l'Authorization Code Flow, con l'aggiunta del code challenge e del code verifier.

- **Implicit Flow**: il suo funzionamento (che presenta somiglianze con il corrispondente grant type OAuth 2.0) consiste in una sola chiamata di autenticazione all'OpenID provider. Da questa chiamata l'applicazione ottiene direttamente un ID token e attraverso essa viene creata o aggiornata la sessione di autenticazione dell'utente. Esporre l'access token in un frammento di URL potrebbe renderlo accessibile tramite la cronologia del browser o nell'header del referer e per questo motivo non è consigliato l'uso di questo tipo di grant per ottenere un access token. L'implicit flow è invece accettabile nel caso in cui un'applicazione debba solo autenticare gli utenti, perché in questo caso può ottenere le informazioni tramite un ID token senza fare uso di un access token. Nel dettaglio, il funzionamento dell'Implicit Flow è il seguente:

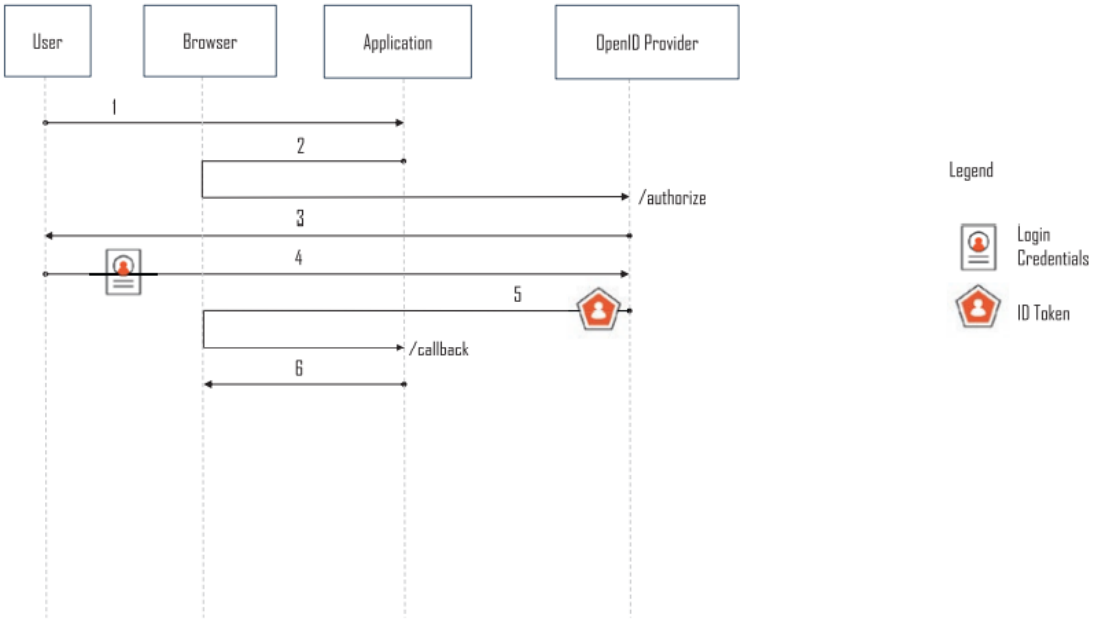

Figura 1.6: Funzionamento dell'Implicit Flow di OpenID Connect[15]

- **Hybrid Flow**: include elementi sia dell'Authorization Code Flow che dell'Implicit Flow. È progettato per applicazioni che utilizzano sia un back-end sicuro, sia un front-end, che utilizza JavaScript, in esecuzione in un browser. Questo flusso effettua due chiamate all'OpenID provider:

  - in seguito all'autenticazione dell'utente avviene la prima chiamata (che risulta in un reindirizzamento verso la parte front-end dell'applicazione), dopo la quale il browser viene reinderizzato all'applicazione con un authorization code e un ID token;

  - la seconda chiamata, effettuata dal back-end, utilizza l'authorization code per ottenere altri token, come un access token ed un refresh token.

  L'Hybrid Flow funziona come mostrato nella seguente immagine:

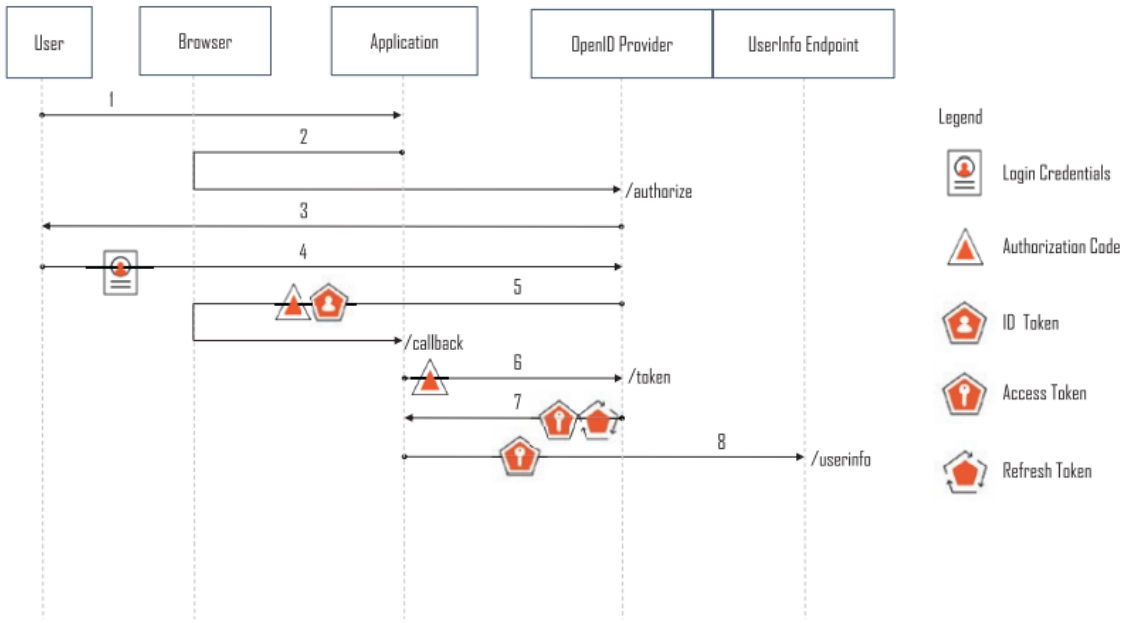

Figura 1.7: Funzionamento dell'Hybrid Flow di OpenID Connect[15]

Con l'Hybrid Flow si possono esporre dei token come con l'Implicit Flow, ma se il token esposto è del tipo ID non si pone il problema. I token che devono rimanere segreti, invece, sono gestiti da un back-end sicuro. Seguendo questo approccio è possibile ottenere contemporaneamente sia un authentication code che dei token di sicurezza.

1.2.3 SAML 2.0

Il Security Assertion Markup Language (SAML) 2.0 è uno standard utilizzato per fornire un'importante funzionalità: il Single Sign-On (SSO) cross-domain. SAML 2.0 è stato adottato in molti ambienti aziendali, in cui è importante avere un unico luogo centrale per la gestione e il controllo delle identità. Infatti, esso consente alle applicazioni utilizzate da dipendenti, clienti e partner di delegare l'autenticazione dell'utente a un identity provider aziendale centralizzato.
Con SAML 2.0, framework basato su XML, un utente può utilizzare le stesse credenziali di accesso per effettuare il login in due o più applicazioni appartenenti a domini differenti. Inoltre, è sufficiente che l'utente acceda ad una singola applicazione che delega l'autenticazione ad un certo identity provider per poter accedere anche alle altre applicazioni senza dover effettuare ulteriori login.

1.2.3.1 Terminologia

Per spiegare il funzionamento di SAML, è necessario specificare alcuni termini che verranno utilizzati:

- **Soggetto**: entità sulla quale verranno scambiate informazioni di sicurezza. Generalmente si riferisce ad una persona, ma può essere una qualsiasi entità capace di autenticarsi.
- **SAML Assertion**: messaggio basato su XML che contiene informazioni di sicurezza riguardanti un soggetto.
- **SAML Profile**: descrive in dettaglio come le assertion, i protocolli e i bindings (i meccanismi utilizzati da SAML per lo scambio di messaggi) di SAML devono essere utilizzati per gestire un certo caso d'uso.
- **Identity Provider**: server che emette delle SAML assertion su un soggetto autenticato, nel contesto del Single Sign-On cross-domain.
- **Service Provider**: elemento che delega l'autenticazione ad un identity provider e che si basa sulle informazioni riguardanti un soggetto autenticato contenute in una SAML assertion, ricevuta dall'identity provider stesso. Il contesto è sempre il Single Sign-On cross-domain.
- **Trust Relationship**: accordo tra un service provider ed un identity provider, in base al quale il primo considera attendibili le assertion emesse dal secondo.
- **SAML Protocol Binding**: descrizione di come gli elementi dei messaggi SAML vengono mappati sui protocolli di comunicazione standard, come HTTP, per la trasmissione tra service provider ed identity provider. In pratica, i messaggi SAML di richiesta e risposta vengono inviati su HTTPS tramite HTTP-Redirect o HTTP-POST, utilizzando rispettivamente i binding HTTP-Redirect e HTTP-POST.

1.2.3.2 Funzionamento

SAML viene usato nel Single Sign-On Web cross-domain, quando il soggetto vuole usare un service provider. Come primo passaggio si ha l'autenticazione del soggetto da parte di un identity provider sotto richiesta dell'applicazione la quale può anche essere in un differente dominio di sicurezza. Dopo l'autenticazione, l'identity provider restituisce all'applicazione la SAML assertion (security token) che contiene informazioni sia sull'evento di autenticazione che sul soggetto. Il service provider e l'identity provider si scambiano metadati (URL degli endpoint, certificati di validazione dei messaggi e altro) per poter eseguire il Single Sign-On Web cross-domain e configurare una trust relationship, senza la quale non può avvenire il processo di autenticazione. Il soggetto

può accedere al service provider solo dopo che i provider hanno eseguito questa vicendevole configurazione, in questo caso il browser sarà reindirizzato all'identity provider a cui viene inviata una richiesta SAML di autenticazione. Dopo l'autenticazione dell'utente, l'applicazione riceve una SAML assertion, contentente le informazioni sul soggetto e sull'evento di autenticazione oppure un errore, nel caso in cui l'operazione non sia andata a buon fine. Nel caso in cui lo stesso soggetto esegua un accesso ad una diversa applicazione che usi, però, lo stesso identity provider, non sarà necessario effettuare una nuova autenticazione in quanto quest'ultimo riconosce l'esistenza di una sessione aperta per l'utente.

1.2.3.3 Authentication Broker

Gli Authentication Broker sono usati per permettere ad una applicazione di supportare differenti protocolli e meccanismi di autenticazione.
Facciamo l'esempio di una applicazione che usi al suo interno OIDC. Nel caso in cui delle aziende che vogliano utilizzare questa applicazione abbiano la necessità di autenticare i propri utenti presso l'identity provider SAML aziendale, diventa utile e consigliabile inserire un authentication broker. In questo modo si evita di implementare SAML all'interno dell'applicazione, un protocollo complesso che aumenterebbe in maniera significativa il lavoro da svolgere e, allo stesso tempo, viene utilizzato un protocollo di autenticazione più recente. Si comunica, quindi, solo col broker tramite OIDC, lasciandogli il compito di far comunicare diversi protocolli di autenticazione su diversi identity provider, permettendo così agli sviluppatori di lavorare sulle funzionalità principali senza disperdere tempo ed energie nell'implementazione di protocolli non necessari, anche se richiesti dai clienti.

1.2.3.4 Identity Federation

L'Identity Federation è una funzionalità di SAML che permette all'identity provider di comunicare con ciascuna applicazione utilizzando un identificatore concordato. Ad esempio, se un'applicazione utilizza come username la mail e un'altra, invece, il nome, l'identity provider può inviare le assertion ai service provider tenendo conto dei differenti identificatori richiesti da ognuno. In caso contrario una delle due applicazioni non riuscirebbe a riconoscere l'utente. Questo accordo sul paramtero da utilizzare può avvenire in fase di configurazione o anche essere richiesto in modo dinamico dal service provider. Senza l'identity federation non sarebbe possibile per le applicazioni utilizzare attributi personalizzati per identificare l'utente.

1.3 Approfondimento: Single Sign-On

La struttura di questa sezione prende ispirazione dal libro [15]. Con il Single Sign-On, attraverso un identity provider, un utente è in grado di accedere a più applicazioni con una sola autenticazione.
Di seguito si prenderanno in considerazione applicazioni che utilizzano OIDC o SAML 2.0, lo stesso identity provider, che sono accessibili con lo stesso browser o, comunque, che usano lo stesso browser per autenticarsi con l'identity provider.
Ci sono diverse situazioni in cui il Single Sign-On risulta particolarmente vantaggioso, come ad esempio in un ambiente rivolto ai consumatori. In questo caso un utente ha spesso la necessità di accedere a diverse applicazioni e risulta quindi utile avere un SSO, come ad esempio il Google Sign-in. Un'altra situazione possibile è il caso di un ambiente aziendale, in cui il dipendente può utilizzare il Single Sign-On per accedere ad applicazioni interne che fanno ricorso all'identity provider aziendale per l'autenticazione. Un terzo scenario potrebbe essere quello universitario, dove studenti, professori e amministratori possono utilizzare il Single Sign-On in tutte le applicazioni sfruttando l'identity provider dell'università.

1.3.1 Vantaggi e svantaggi

Il Single Sign-On offre molti vantaggi. In primo luogo, per l'utente c'è la comodità di non dover eseguire troppo spesso l'autenticazione, evitando così di comunicare continuamente le proprie credenziali alle applicazioni. Dal punto di vista dello sviluppatore, invece, c'è la possibilità di delegare all'identity provider diverse attività, come l'onere dell'implementazione delle pagine di accesso, della convalida e archiviazione sicura delle credenziali e di alcune funzionalità di ripristino dell'account. L'azienda o organizzazione che lo utilizza, inoltre, ottiene il vantaggio di avere un unico luogo in cui implementare e applicare i criteri di autenticazione e le loro diverse forme, e gestire la registrazione, il ripristino e la chiusura dell'account. Anche la sicurezza trae vantaggio dall'uso del SSO, perché quando l'utente ha una sola password da memorizzare diminuisce la probabilità di appuntarsela in luoghi poco sicuri, come foglietti o file senza protezione.
L'utilizzo del SSO comporta anche alcune problematiche: la sua implementazione potrebbe creare un single point of failure, ed un servizio centralizzato fornisce anche un unico punto di attacco. Queste problematiche possono essere parzialmente superate selezionando un identity provider progettato per avere un'high avaliability e prestare molta attenzione alla sicurezza. Resta comunque importante anche tenere in considerazione il fatto che l'identity provider ha una piena visibilità dell'attività degli utenti e la capacità di tenere traccia dei siti che visitano, fattori che potrebbero essere un problema per la privacy. In sintesi, quindi, quando utenti e aziende selezionano un identity provider de-

vono eseguire un'attenta valutazione di come viene gestita la privacy e delle certificazioni di sicurezza del provider, prima di affidargli l'autenticazione della loro applicazione.

1.3.2 Funzionamento

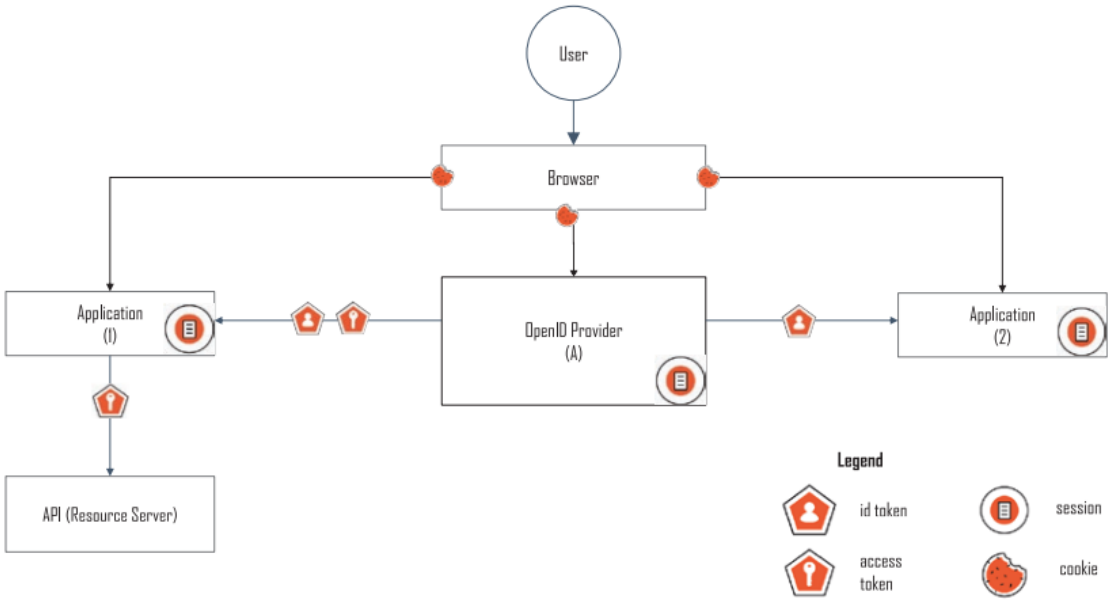

Figura 1.8: Funzionamento del Single Sign-On[15]

Il punto di inizio è la richiesta di accesso di un utente ad un'applicazione. Questa, attraverso il proprio browser, manda una richiesta di autenticazione all'identity provider il quale, dopo che l'utente si è autenticato con successo, apre una sessione creando un cookie nel browser con le informazioni su di essa. Quindi, lo stesso identity provider invia i security token (che contengono i dati sia sull'utente che sull'evento di autenticazione) all'applicazione che può in questo modo creare o aggiornare la propria sessione locale ed eventualmente un cookie per l'utente. Dopo tutti questi passaggi, lo stesso utente utilizzando lo stesso browser potrebbe chiedere di accedere ad una seconda applicazione. Questa, rilevando che l'utente non ha ancora effettuato l'accesso, si rivolge all'identity provider inviandogli il cookie creato dall'identity provider stesso e salvato sul browser. In questo modo, l'identity provider riesce a individuare la sessione già attiva e verifica se è valida e, in caso affermativo, invia alla seconda applicazione i security token senza richiedere un nuovo inserimento delle credenziali. A questo punto anche la seconda applicazione, come la prima, creerà o aggiornerà una sessione locale per l'utente. Questo, poi, potrà accedere ad altre applicazioni o alle due precedenti senza dover inserire le

proprie credenziali fino a che la sessione presso l'identity provider (SSO session) resterà valida.
I motivi che invalidano la sessione di un utente sono diversi: potrebbe essere scaduta, oppure terminata da un amministratore dell'identity provider o ancora l'utente potrebbe essersi scollegato dal provider. Anche la disconnessione dell'utente da un'altra applicazione può terminare la sessione dell'identity provider. In ogni caso, indipendentemente dal motivo, ad un utente verrà chiesto di autenticarsi nuovamente nel caso in cui la richiesta di autenticazione raggiunga l'identity provider quando la sua sessione non è più valida.
Ci sono situazioni in cui è necessario che l'utente interagisca con l'identity provider anche quando la sessione è ancora valida:

- l'applicazione a cui accede l'utente ha bisogno di richiamare delle API, sempre per conto dell'utente, e l'identity provider è anche il server di autorizzazione. In questo caso sarà lo stesso identity provider a richiedere all'utente il consenso per accedere a queste API
- l'utente accede ad una applicazione che necessita di una autenticazione più forte o comunque differente da quella attuale, di conseguenza dovrà soddisfare i nuovi requisiti
- a volte, per motivi di sicurezza, l'applicazione stessa include nella richiesta all'identity provider un parametro che forza l'autenticazione anche in caso di sessione valida

Se non ci sono casi particolari, però, il SSO consente sempre di accedere a più applicazioni con una sola autenticazione fino alla scadenza della sessione.

CAPITOLO 2

Keycloak

2.1 Panoramica

Questa introduzione generale a Keycloak riprende i concetti espressi nel libro [14].
Keycloak è un progetto open source nato nel 2014 che aveva come obiettivo principale quello di aiutare gli sviluppatori nel compito di proteggere le proprie applicazioni. Da allora si è sviluppato, grazie a una comunità forte, come uno strumento di gestione dell'identità e degli accessi che pone particolare attenzione alle API REST e alle applicazioni moderne, come single-page application o applicazioni mobili. Viene utilizzato in situazioni molto differenti tra loro, che variano da piccoli siti Web con pochi utenti a grandi aziende con milioni di utenti.
Tra le altre cose, Keycloak consente di utilizzare temi per personalizzare le sue pagine visibili dall'utente, oltre a dei metodi per customizzare la pagina di login, ad esempio includendo flussi come il recupero della password, la richiesta di aggiornarla con una certa frequenza e l'accettazione di termini e condizioni. È inoltre possibile configurare metodi di autenticazione più forti. Grazie a queste caratteristiche risulta semplice integrare le pagine con un eventuale marchio aziendale o un'applicazione esistente.
L'utilizzo di Keycloak porta diversi vantaggi, sia per un applicazione che gli delega l'autenticazione, sia per la sicurezza. Nel primo caso il beneficio deriva dal fatto che l'applicazione non debba preoccuparsi di implementare i meccanismi di autenticazione ne di memorizzare i dati e le password in modo sicuro. La sicurezza, invece, aumenta di livello poiché le applicazioni non accedono direttamente alle credenziali dell'utente.
Tramite Keycloak gli utenti possono accedere a più applicazioni attraverso una sola autenticazione perché permette di utilizzare un Single Sign-On e avere un meccanismo di gestione delle sessioni. Esso, inoltre, consente ad amministratori e utenti di vedere a quali applicazioni sono autenticati ed eventualmente terminare le sessioni in remoto.
I protocolli supportati sono quelli standard: Keycloak supporta OAuth 2.0, OpenID Connect e SAML 2.0, facilitando così l'integrazione sia con applicazioni nuove che con

quelle già esistenti.
I dati riguardanti gli utenti di Keycloak possono essere memorizzati in un database interno ad esso, il che facilita l'approccio iniziale. Un'altra opzione è utilizzare un'infrastuttura di identità esistente, ottenendo le informazioni da essa. Tale infrastuttura potrebbe essere un database di utenti di un certo social network o di un identity provider aziendale. Infine, può avvenire l'integrazione con delle directory utente esistenti, quali Active Directory e LDAP.
Keycloak è leggero, facile da installare, altamente scalabile e fornisce un'high avaliability. È fortemente personalizzabile ed estendibile, ma allo stesso tempo è abbastanza semplice e di uso immediato, grazie al supporto per casi di uso comune. Tra le estensioni che si possono applicare a Keycloak ci sono l'integrazione con archivi customizzati di utenti, la creazione ed utilizzo di meccanismi di autenticazione personalizzati, la manipolazione dei token e l'utilizzo di protocolli di login personalizzati.
Le funzionalità di Keycloak verranno approfondite in una sezione successiva.
Le informazioni presentate nelle prossime due sezioni sono ricavate principalmente dalla documentazione di Keycloak [9][6].

2.2 Concetti principali

Prima di spiegare più dettagliatamente le funzionalità di Keycloak, è necessario definire alcuni concetti:

- **Realm**, è un concetto in Keycloak che si riferisce ad un oggetto che gestisce un insieme di utenti con le loro credenziali, ruoli e gruppi. Si possono avere più realm in un server Keycloak, i quali saranno indipendenti l'uno dall'altro ed ognuno gestirà solo i propri utenti. Ogni realm è quindi completamente isolato dagli altri, ha una propria configurazione ed un proprio insieme di applicazioni ed utenti. Ciò consente di utilizzare una singola installazione di Keycloak per molteplici scopi. Ad esempio, si potrbbe voler avere un realm per le applicazioni e i dipendenti interni e un altro per le applicazioni e i clienti esterni. Esiste un tipo particolare di realm, il master realm. È al livello più alto nella gerarchia dei realm e gli account di amministrazione in esso possono visualizzare e gestire qualsiasi altro realm creato nell'istanza del server. Il master realm viene creato alla prima installazione di Keycloak ed è consigliato utilizzarlo solo per creare e gestire altri realm, e non per gestire utenti ed applicazioni.

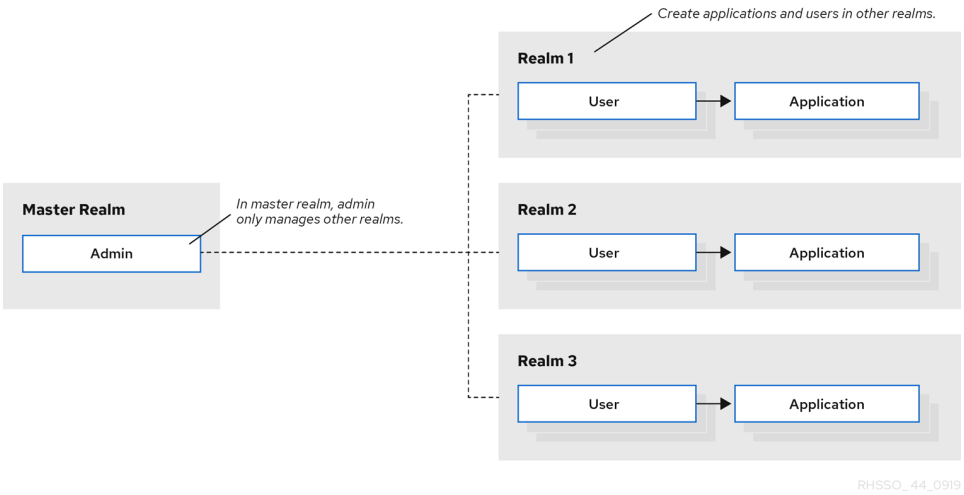

Figura 2.1: Master realm di Keycloak[11]

- **Client**, è un concetto simile a quello definito nella sezione su OAuth 2.0. In sostanza, è un'entità che può richiedere a Keycloak di autenticare un utente. Di solito un client è un'applicazione o un servizio che vuole utilizzare Keycloak per fornire il servizio di Single Sign-On, ma potrebbe anche essere un'entità che vuole solo ottenere informazioni di identità o un access token per invocare altri servizi che utilizzano Keycloak.

- **Ruoli**, identificano un tipo o una categoria di utenti. Amministratore, utente, manager e dipendente sono ruoli tipici che possono esistere in un'organizzazione. Le applicazioni spesso assegnano le autorizzazioni a ruoli specifici piuttosto che a singoli utenti, poiché la gestione dei singoli utenti potrebbe essere troppo complessa. Tramite l'admin console di Keycloak è possibile creare tali ruoli, scegliendo se ognuno appartiene all'intero realm considerato o ad un singolo client in esso.

- **Mapping tra utenti e ruoli**, definisce una mappatura tra un ruolo e un utente. Un utente può essere associato a zero o più ruoli. Queste informazioni sui ruoli associati agli utenti possono essere incapsulate in token e assertion in modo che le applicazioni possano decidere i permessi di accesso alle varie risorse che gestiscono. Anche in questo caso, Keycloak fornisce un modo per effettuare il mapping tra ruoli ed utenti.

- **Scope**, un concetto già trattato nelle sezioni riguardanti i protocolli. Viene utilizzato in OAuth 2.0 ed in OIDC negli access token e negli ID token. Nella richiesta di un access token lo scope definisce ciò a cui l'applicazione ha bisogno di accedere e a cui bisogna dare il consenso, mentre in una richiesta di autenticazione può essere

usato per specificare l'insieme delle informazioni che si vogliono ottenere dall'ID token.

- **Gruppi**, servono per la gestione di gruppi di utenti. Ad un gruppo possono essere associati una serie di attributi e ruoli. Gli utenti che diventano membri di un gruppo ereditano gli attributi e il mapping dei ruoli che tale gruppo definisce. Keycloak fornisce un metodo per creare dei gruppi ed associarli agli utenti.
- **Client adapter**, sono plugin da installare in un'applicazione per permetterle di uilizzare Keycloak e comunicare con esso. Keycloak ha una serie di adapter che possono essere scaricati per diverse piattaforme, inoltre esistono adapter di terze parti che è possibile utilizzare. In sostanza, sono librerie che semplificano la protezione di applicazioni e servizi con Keycloak. Vengono chiamati adapter piuttosto che librerie in quanto forniscono una stretta integrazione con la piattaforma ed il framework sottostanti. Ciò rende gli adapter facili da usare e richiedono meno codice boilerplate rispetto a quanto richiesto in genere da una libreria.
- **Protocol mapper**, in ogni client è possibile personalizzare quali claim vengono scritti nel token OIDC o nella SAML assertion. Per farlo, Keycloak permette di creare e configurare i protocol mapper nei diversi client.

2.3 Funzionalità principali

Keycloak fornisce numerose funzionalità, le più importanti sono quelle mostrate in Fig.2.2 e vengono spiegate di seguito.

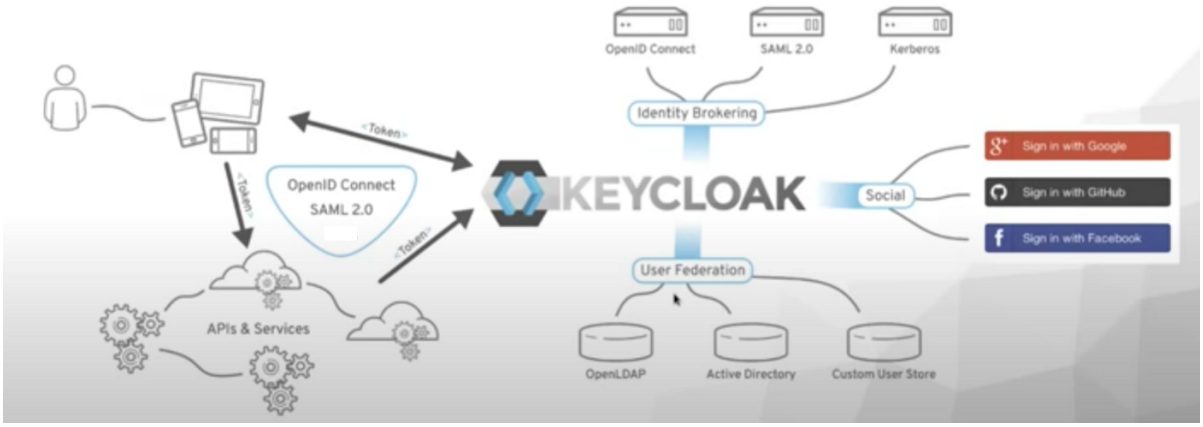

Figura 2.2: Funzionalità più importanti di Keycloak[13]

- **Single-Sign On e Single-Sign Out per applicazioni web**, tale concetto è già stato spiegato in precedenza. Riassumendo, gli utenti si autenticano con Keycloak piuttosto che con le singole applicazioni. Ciò significa che le applicazioni non devono gestire l'accesso, l'autenticazione degli utenti e l'archiviazione degli utenti. Inoltre, una volta effettuata l'autenticazione a Keycloak, gli utenti non devono effettuarla nuovamente per accedere ad un'applicazione diversa. Ciò può essere applicato anche al logout: Keycloak fornisce il Single-Sign Out, il che significa che gli utenti devono disconnettersi una sola volta per disconnettersi da tutte le applicazioni che utilizzano Keycloak.

- **Supporto per OAuth 2.0**, Keycloak può utilizzare il protocollo OAuth 2.0 per autorizzare un'applicazione ad accedere a risorse di applicazioni diverse da essa.

- **Supporto per OpenID Connect**, Keycloak supporta anche il protoccollo OIDC che, come già visto, è un protocollo di autenticazione che estende OAuth 2.0.

  Esistono diversi endpoint OIDC che possono essere utilizzati con Keycloak; questi URL sono utili in particolare nel caso in cui ci sia la necessità di utilizzare un client adapter non fornito da Keycloak per comunicare tramite OIDC con, appunto, Keycloak. Di seguito venogno riportati gli URL relativi di tali endpoint; la loro radice è composta dal prtocollo (http o https), il dominio o nome dell'host ed */auth*. Un esempio di radice è quindi https://aminsep.disi.unibo.it/auth.

  Innanzitutto, l'endpoint principale è quello di configurazione, che è come una directory root. Esso restituisce, infatti, tutti gli altri endpoint con alcune informazioni legate ad essi, ed il suo URL è **/realms/realm-name/.well-known/openid-configuration**. Di seguito, i principali endpoint:

  /realms/{realm-name}/protocol/openid-connect/auth serve per ottenere un codice temporaneo nell'Authorization Code Flow o per ottenere un token utilizzando l'Implicit Flow o l'Hybrid Flow.

  /realms/{realm-name}/protocol/openid-connect/token utilizzato nell'Authorization Code Flow per trasformare un codice temporaneo in un token o per ottenere i token direttamente tramite il Resource Owner Password Credentials Grant o il Client Credentials Grant.

  /realms/{realm-name}/protocol/openid-connect/logout l'URL per effettuare il logout.

  /realms/{realm-name}/protocol/openid-connect/userinfo l'endpoint per le informazioni sull'utente, descritto in precedenza.

  In ognuno degli URL, {realm-name} va sostituito con il nome del realm considerato.

- **Supporto per SAML**, tramite Keycloak è possibile anche utilizzare SAML 2.0. Come già detto, SAML definisce diversi modi per scambiare documenti XML durante la sua esecuzione, chiamati *binding*. I tipi di binding che Keycloak supporta sono il Redirect ed il Post per le applicazioni web e l'ECP per le chiamate REST.

  Keycloak ha un singolo endpoint per SAML, usato da tutti i binding: **http(s)://authserver.host/auth/realms/{realm-name}/protocol/saml**.

- **Identity Brokering**, il compito di un'identity broker è quello di fare da intermediario fra diversi service provider ed identity provider. In questo modo un'applicazione può interagire con esso utilizzando un singolo metodo, dopo di che è l'identity broker che comunica coi diversi identity provider. L'identity broker ha anche il compito di instaurare una relazione di fiducia con gli identity provider dei quali vuole utilizzare le identità.

  Quando viene utilizzato Keycloak come identity broker, un elenco di identity provider ai quali è possibile autenticarsi viene mostrato all'utente che ne dovrà selezionare uno.

  In sostanza, il flusso che segue un utente per autenticarsi con un identity broker è mostrato in Fig.2.3.

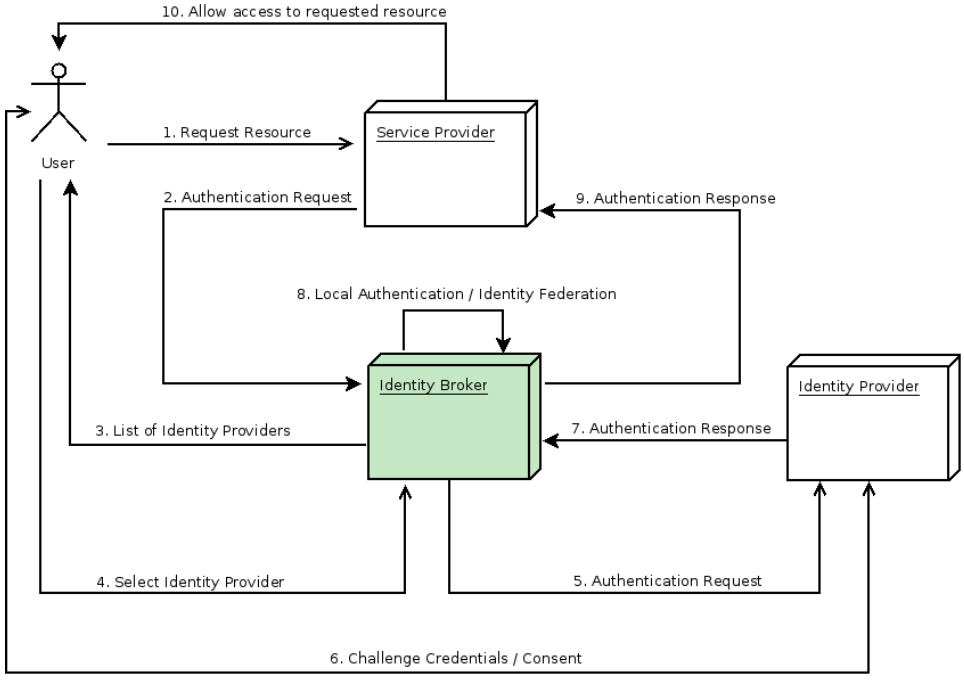

Figura 2.3: Identity Brokering in Keycloak[10]

Il flusso inizia con un service provider che riceve da parte di un utente una richiesta di accesso a risorse protette. Come conseguenza, il browser viene reinderizzato a Keycloak al quale viene presentata una richiesta di autenticazione. Keycloak mostra un elenco di identity provider e l'utente ne seleziona uno, quello al quale vuole autenticarsi. Viene quindi reinderizzato con una richiesta di autenticazione alla pagina di login dell'identity provider, nella quale inserisce le proprie credenziali e, se sono corrette, torna a Keycloak con una risposta che contiene un security token. A questo punto, Keycloak controlla che la risposta sia valida e se lo è crea localmente un nuovo utente, nel caso in cui tale utente non sia già esistente. Alla fine di questo passaggio, Keycloak autentica l'utente e crea il proprio token; in questo modo si ha l'autenticazione locale. Questo token viene poi restituito al service provider durante il reindirizzamento dell'utente ad esso. Infine, quindi, il service provider permette all'utente di accedere alle risorse richieste inizialmente.

- **Social Login**, solitamente, ogni volta che un utente vuole usare una nuova applicazione deve registrarsi e perciò utilizzare un nuovo nome utente e password. Per una persona può diventare oneroso ricordare un gran numero di nomi utente e password diversi. Per diminuire le problematiche di questa situazione possono essere usati gli identity provider social. Tramite questi ultimi, è possibile delegare l'autenticazione ad un'entità semi-attendibile e rispettata in cui l'utente probabilmente ha già un account. Keycloak permette di autenticarsi tramite diversi social network come Facebook, Microsoft, LinkedIn, StackOverflow, Github, Google e Twitter. Questa sua proprietà viene chiamata Social Login.

- **User Storage Federation**, una funzionalità utilizzata nel caso in cui un certo client disponga già di un qualche tipo di database di utenti e voglia utilizzare le informazioni presenti in esso per l'autenticazione. Per permettere ciò, Keycloak fornisce un metodo per effettuare la sincronizzazione con tale database. LDAP e Active Directory sono supportati da Keycloak senza bisogno di estenderlo. Esiste anche la possibilità di creare estensioni personalizzate per qualsiasi database utente. Per aggiungere questa funzionalità a Keycloak bisogna configurare uno User Storage provider per ogni database esterno che si vuole utilizzare.

  Quando Keycloak riceve la richiesta di autenticazione di un utente, deve controllare il suo database interno per verificare la presenza di tale utente. In caso negativo, dovrà controllare tutti gli storage provider che sono stati configurati fino a che non riuscirà a trovare l'utente cercato. Keycloak utilizzerà poi un modello comune allo storage provider, nel quale vengono mappati i dati dell'utente, per ottenere le informaioni da inserire in un token OIDC o una SAML assertion.

  La necessità di memorizzare informazioni dell'utente in Keycloak potrebbe esserci anche nel caso in cui si usi lo User Federation. Questo avviene di solito se i dati

nel database non sono sufficienti per supportare tutte le funzionalità di Keycloak, come ad esempio quando l'archivio esterno di utenti non supporta OTP.

- **Admin Console**, è una GUI attraverso la quale gli amministratori possono gestire in modo centralizzato tutti gli aspetti del server Keycloak. Tra le operazioni che si possono effettuare tramite l'Admin Console ci sono: abilitare e disabilitare diverse funzionalità, configurare l'identity brokering e l'user federation, creare e gestire i client, definire criteri di autorizzazione granulari e gestire gli utenti ed i loro dati, inclusi i loro permessi e le loro sessioni, oltre a molte altre cose.

  La pagina principale della console di Keycloak è la seguente:

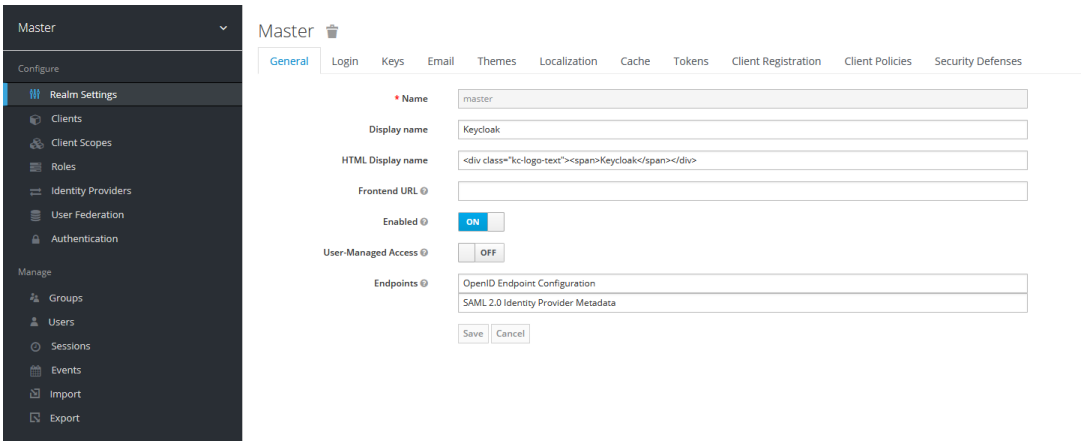

Figura 2.4: Admin console di Keycloak

  A sinistra si possono notare le varie sezioni attraverso le quali si può accedere alle varie funzioni.

- **Account Console**, attraverso tale console, gli utenti possono gestire in modo centralizzato i propri account. Le funzioni principali che si possono utilizzare sono l'aggiornamento del profilo, la modifica delle password e l'impostazione dell'autenticazione a due fattori. Gli utenti possono anche gestire le proprie sessioni e visualizzare la cronologia delle azioni dell'account. Se il social login o l'identity brokering sono abilitati, un utente può anche collegare i propri account a provider aggiuntivi.

  È possibile accedere a tale console tramite il seguente URL relativo: **/auth/realms/{realm-name}/account** ed il suo aspetto viene mostrato in figura.

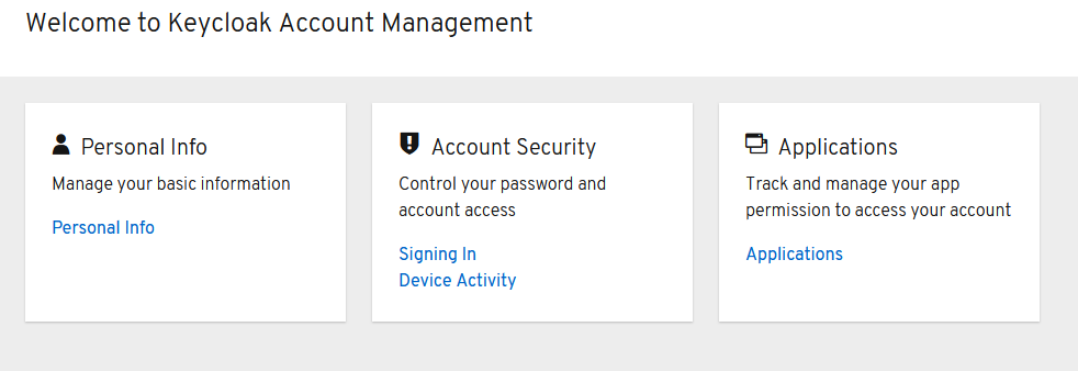

Figura 2.5: Account console di Keycloak

Da questa schermata è possoibile accedere alle varie funzionalità.

Altre funzionalità di Keycloak degne di nota sono le seguenti:

- **Autenticazione a due fattori**, Keycloak supporta TOTP/HOTP tramite Google Authenticator o FreeOTP.
- **Gestione delle sessioni**, Keycloak crea e mantiene una sessione per ogni singolo utente che accede ad un realm e ricorda tutti i client che l'utente visita durante questa sessione. L'Admin Console permette all'amministratore di effettuare diverse operazioni relative alle sessioni, come ad esempio disconnettere uno o più utenti, oppure revocare un token o ancora impostare il timeout di sessione. Inoltre, Keycloak utilizza le informazioni estrapolate dal tracciamento delle operazioni degli utenti per fornire all'amministratore statistiche di login riguardanti l'intero realm e permettergli di controllare per ogni client chi ha effettuato l'accesso e dove.
- **Token mapper**, Keycloak permette di mappare attributi utente, ruoli, ecc. in token e statement in modo personalizzato.
- **Service Provider Interfaces (SPI)**, nella maggioranza dei casi, Keycloak non ha bisogno di richiedere codice personalizzato, ma quando questo si rende necessario Keycloak utilizza una serie di Service Provider Interface per consentire la personalizzazione di vari aspetti del server, come ad esempio il flusso di autenticazione, lo User Federation Provider e il Protocol Mapper, senza la necessità di modificare il sorgente.
- **Supporto di piattaforme e linguaggi che hanno librerie OpenID Resource Provider o SAML Service Provider**, Keycloak fornisce i propri adapter per alcune piattaforme selezionate, ma è anche possibile utilizzare librerie generiche OpenID Connect Resource Provider e SAML Service Provider.

- **Servizi di autorizzazione**, Keycloak supporta l'applicazione di policy di autorizzazione in maniera granulare ed è in grado di combinare diversi meccanismi di access control come:
  - Attribute-based access control (ABAC)
  - Role-based access control (RBAC)
  - User-based access control (UBAC)
  - Context-based access control (CBAC)
  - Rule-based access control
  - Time-based access control
  - Meccanismi di access control personalizzati tramite una Service Provider Interface (SPI)

  L'accesso ad una risorsa protetta viene concesso dai resource server in base ad alcune specifiche informazioni. Se i resource server si basano su RESTful, ad esempio, allora sono i security token, inviati come bearer token ad ogni singola richiesta al server, a fornire queste informazioni. Quando, invece, le applicazioni web utilizzano le sessioni per l'autenticazione dell'utente, sono le sessioni stesse che contengono le informazioni necessarie così da recuperarle ad ogni richiesta. Per migliorare le capacità di autorizzazione delle applicazioni, Keycloak mette a disposizione diversi servizi come criteri di autorizzazione granulari e diversi meccanismi di access control per proteggere le risorse, la gestione centralizzata di policy, risorse e permessi e un punto di decisione centralizzata delle policy.

CAPITOLO 3

Compositional Agile System

Nonostante il manifesto agile [1] ponga l'accento sugli individui e la loro interazione con processi e strumenti, è oggi importante concentrarsi anche sugli strumenti in se, utili per supportare diversi metodi di sviluppo. In particolare, nello sviluppo di software complessi ci sono varie attività specifiche che possono essere automatizzate, diventando in questo modo anche più strutturate. In un contesto in cui la metodologia agile sta velocemente prendendo piede, è importante avere dei tool che permettano di semplificare il lavoro di uno sviluppatore e facilitare l'utilizzo delle pratiche agili.
CAS[4], ovvero Compositional Agile System, è un ambiente creato per supportare l'utilizzo di iAgile, un modello agile derivato da Scrum, ideato per sviluppare sistemi critici in ambito militare. Tutto il software presente in CAS è open source e l'ambiente può essere messo in opera su una rete privata o in un cloud ibrido.

3.1 Struttura

La struttura di CAS è mostrata in Fig.3.1. CAS è diviso in due parti: CAS Client e CAS Server, spiegate nel dettaglio di seguito. Il CAS considerato in questo capitolo è CAS 2.0, ottenuto da una reingegnerizzazione della versione 1.0.

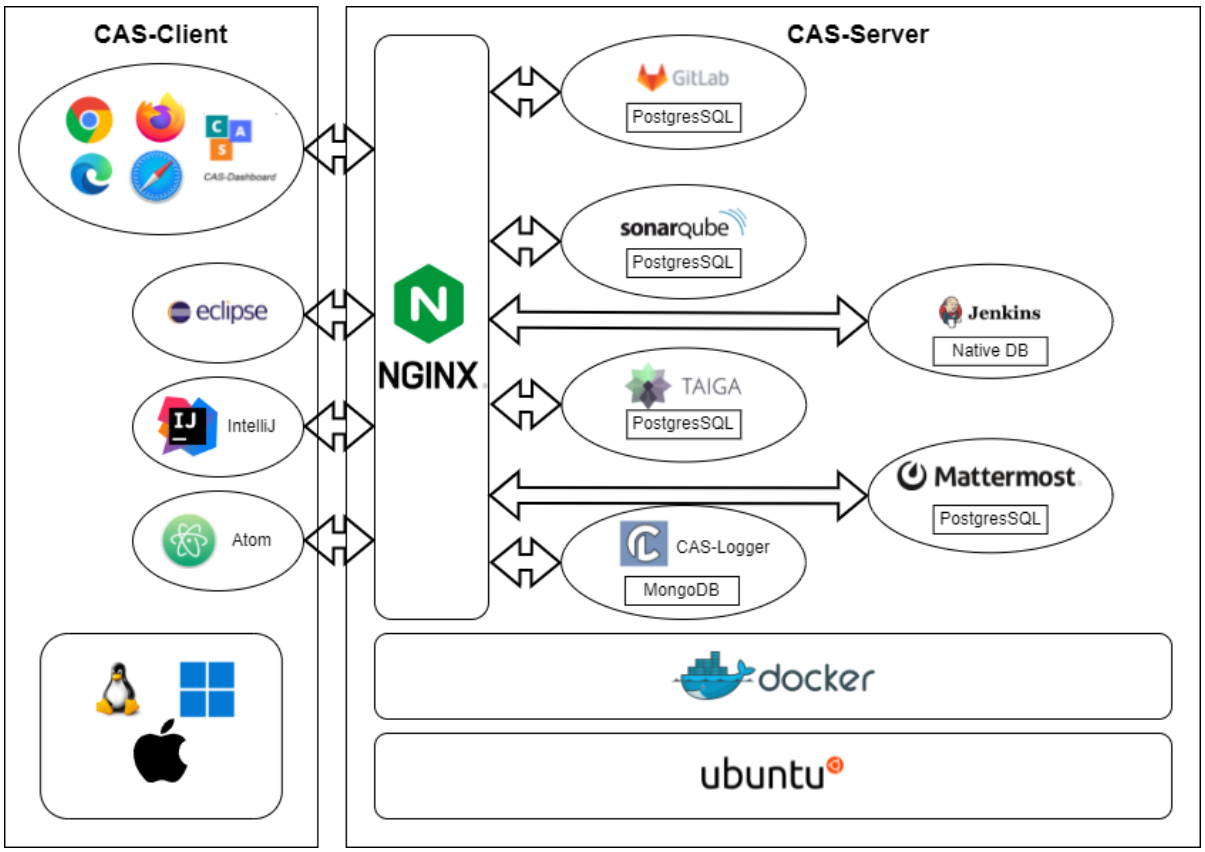

Figura 3.1: Struttura di CAS 2.0

3.1.1 CAS Server

La parte server di CAS comprende diversi servizi ed è completamente dockerizzata. Dalla versione 2.0 è possibile gestire le configurazioni di ogni servizio tramite un singolo file, quello utilizzato dal tool *compose* di docker. Come si vede dall'immagine, viene anche utilizzato Nginx per svolgere il ruolo di reverse proxy.
Un reverse proxy serve per smistare il traffico proveniente dall'esterno tra i servizi disponibili e viene utilizzato con Docker come mostrato in Figura 3.2:

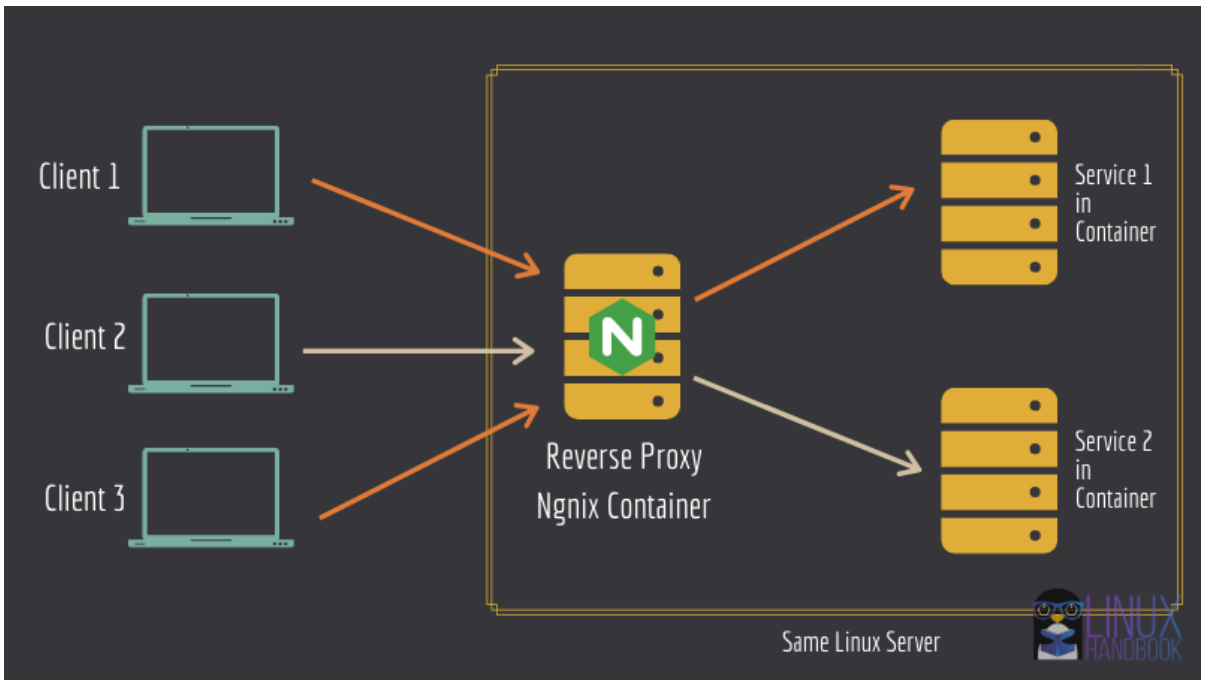

Figura 3.2: Reverse proxy con Docker e Nginx[3]

Più nel dettaglio, diversi client effettuano richieste al server che verrano ricevute dal reverse proxy. Nginx poi, in base all'URL della richiesta, la passerà al servizio corretto. Utilizzando questo metodo non è necessario che gli utenti specifichino i diversi numeri di porta per i servizi desiderati, ai quali possono accedere semplicemente tramite URL definibili. Per la comunicazione interna al server viene creata una rete isolata alla quale appartengono i diversi container Docker, la quale può non utilizzare HTTPS non essendo esposta all'esterno. Un'ulteriore vantaggio è che evantuali certificati SSL possono essere gestiti in un singolo punto, ovvero Nginx, essendo l'unico che interagisce con la rete esterna.
Di seguito vengono spiegati nel dettaglio tutti gli elementi compresi in CAS Server.

Docker

Docker è un progetto open source sviluppato dalla Docker, Inc. ed è stato presentato al pubblico per la prima volta nel 2013. Docker è una piattaforma usata per la creazione, l'implementazione e la gestione di applicazioni all'interno di container, dei componenti standardizzati che impacchettano il software fornendo tutto ciò che gli serve per essere eseguito, come librerie e dipendenze del sistema operativo. Ogni container è indipendente dagli altri ed ha il proprio software, le proprie libreire e i propri file di configurazione. La comunicazione tra container avviene tramite canali appositi e ben definti. Docker utilizza la virtualizzazione a livello di sistema operativo e siccome tutti i container condividono i

servizi di un singolo kernel, utilizzano meno risorse rispetto alle macchine virtuali. Ogni container, quindi, utilizza un proprio sistema operativo e può essere eseguito su Linux, Windows e macOS.
Gli elementi principali di Docker sono il *daemon*, un processo che gestisce i container e fornisce una CLI che permette agli utenti di interagire con l'implementazione di Docker; i container, già spiegati; le immagini, dei modelli di sola lettura utilizzati per creare container ed i registri; dei repository per salvare e distribuire immagini Docker. I due registri pubblici principali sono Docker Hub, utilizzato di default, e Docker Cloud.
Infine, in CAS viene utilizzato il tool Docker Compose. Questo strumento serve per definire ed eseguire applicazioni multi-container. Docker Compose crea un file YAML che serve per gestire l'architettura dell'applicazione, specificando quali servizi sono inclusi nell'applicazione e come devono essere configurati. Con questo tool è possibile implementare ed eseguire i container con un singolo comando.

Nginx

Nginx è un server web open source leggero e ad alte prestazioni. Viene usato anche come reverse proxy, load balancer per server HTTP, TCP e UDP, proxy di posta elettronica e cache HTTP. Attualmente, numerosi web server usano Nginx, soprattutto come load balancer. Questo software è distribuito sotto licenza BSD-like e può essere utilizzato su vari sistemi operativi, tra i quali Unix, Linux, macOS e Microsoft Windows. Nginx fornisce rapidamente i contenuti statici con un uso efficiente delle risorse di sistema e, nella gestione delle richieste, utilizza un approccio asincrono basato sugli eventi in modo da ottenere prestazioni migliori sotto stress, in contrasto con il modello del server HTTP Apache che usa un approccio orientato ai thread o ai processi.
Nginx è stato scritto per risolvere la difficoltà di gestire numerose connessioni concorrenti, problema comune tra i web server. La prima versione è stata rilasciata nel 2004 e si è diffuso velocemente. Nel 2011 è stata fondata l'azienda NGINX, Inc. per supportare lo sviluppo di Nginx e per commercializzare Nginx Plus, un prodotto con funzionalità aggiuntive progettato per il mercato aziendale. Nel 2019 la società è stata acquisita dalla F5 Networks, Inc.
Ad oggi, Nginx e Nginx Plus possono gestire centinaia di migliaia di connessioni simultanee e sono diventati altamente popolari. Un sondaggio del 2018 sull'utilizzo di Docker ha inoltre rilevato che Nginx era la tecnologia più comunemente utilizzata nei container Docker.

CAS Logger

CAS include un servizio di self tracking che può essere utilizzato da uno sviluppatore per tenere traccia delle azioni effettuate durante il processo di scrittura del codice su un IDE, che deve essere tra quelli per i quali è presente un plugin nel CAS Client. I dati

aggregati relativi ad un utente possono, poi, essere visualizzati sulla CAS dashboard, alla quale si accede tramite browser.
Nella versione precedente di CAS il logger utilizzava un server esterno [7]. Nella versione 2.0 si è provveduto ad effettuare un fork per portarlo sotto il controllo diretto dei team. Il logger è presente sul repository di GitHub[8].

Jenkins

Jenkins è un software open source scritto in Java. È utilizzato per supportare gli sviluppatori, aiutandoli ad automatizzare diversi aspetti dello sviluppo software relativi alle parti di build, di test e di distribuzione. Questo strumento facilita l'utilizzo della CI/CD, ovvero continuous integration e continuous delivery, ad oggi sempre più diffusa. Questa pratica è anche alla base della metodologia DevOps, che prevede sviluppo continuo, test continui, integrazione continua, distribuzione continua e monitoraggio continuo delle applicazioni durante tutto il ciclo di vita dello sviluppo del software.
Jenkins può essere usato con i principali strumenti di gestione del codice sorgente, come Git Concurrent Versions System, Subversion, Mercurial e Perforce. Può inoltre eseguire progetti scritti in Ant o Maven e script bash o comandi batch di Windows.
L'esecuzione di alcune attività che Jenkins automatizza è programmabile con una pianificazione simile a quella di *cron* o utilizzando un trigger, ad esempio il commit del proprio codice all'interno di un repository potrebbe causare il lancio di test, i cui risultati verranno analizzati per avere un feedback immediato sulla sua qualità.
Le funzionalità di Jenkins possono anche esssere potenziate dal grande numero di plugin disponibili, sviluppati e diffusi dalla comunità di utenti e sviluppatori che lo usa. Essi possono avere diversi scopi quali l'integrazione con strumenti di controllo delle versioni, grandi basi di dati e tool di build.
Il progetto Jenkins è nato nel 2004 con il nome di Hudson, ma nel 2011 la community l'ha ribattezzato Jenkins a seguito di una disputa con Oracle.

SonarQube

SonarQube è una piattaforma open source disponibile sia nella versione gratuita sotto licenza GNU Lesser General Public, sia in quella enterprise con licenza a pagamanento. Sviluppata da SonarSource, permette l'ispezione continua della qualità del codice eseguendo analisi statiche su di esso dalle quali ricavare dei report su vulnerabilità di sicurezza, duplicazione del codice, code smells, bug, standard di codifica e molto altro. I linguaggi di programmazione supportati sono diversi, ad esempio Java, C#, C, C++, JavaScript, Python, Swift, PHP, HTML, CSS, PL/SQL e XML. Alcuni dei linguaggi che supporta sono disponibili solo nella versione commerciale. SonarQube può anche registrare la cronologia delle metriche e fornire grafici di evoluzione. SonarQube fornisce, inoltre, analisi e integrazioni completamente automatizzate con Maven, Ant, Gradle,

MSBuild e diversi strumenti di integrazione continua come Jenkins. Infine, Sonarqube può essere integrato con Eclipse, Visual Studio, Visual Studio Code, e IntelliJ IDEA tramite il plugin SonarLint.

GitLab

GitLab è una piattaforma web open source il cui principale compito è gestire i repository Git, ma fornisce anche diverse altre funzionalità. GitLab è un sistema di controllo di versione, quindi pertmette la creazione di repository pubblici o privati tramite i quali è possibile controllare le diverse verioni del software. GitLab, inoltre, permette a più persone di lavorare ad uno stesso progetto parallelamente senza generare conflitti. Questa piattaforma fornisce anche funzionalità come issue-tracking, wiki e un sistema di Continuous Integration. Appartenente alla GitLab Inc., è disponibile sia nella versione gratuita che in quella a pagamento e da ciò dipende quali sono le funzionalità supportate. Il linguaggio in cui era scritto originariamente GitLab è Ruby, a cui succesivamente si sono aggiunte parti in Go. Il progetto iniziale prevedeva che GitLab fosse una soluzione di gestione del codice sorgente per collaborare all'interno di un team, ma nelle versioni successive si è evoluto in una soluzione integrata che copre l'intero ciclo di vita dello sviluppo del software. L'attuale stack tecnologico include Go, Ruby on Rails e Vue.js.
GitLab è stato fondato da Dmitriy Zaporozhets e Valery Sizov nell'ottobre 2011 e viene distribuito con licenza MIT.

Taiga

Taiga è un sistema di gestione dei progetti gratuito ed open source per startup, sviluppatori agili e designer. È diviso in una parte frontend ed una backend, la prima scritta in AngularJS e CoffeeScript, e la seconda in Django e Python. Taiga tiene traccia dello stato di avanzamento di un progetto e possono essere usati il modello Kanban, Scrum, o entrambi. I backlog dei progetti vengono mostrati come un elenco di tutte le funzionalità e le User Story aggiunte ad esso. Taiga può interfacciarsi con i repository di controllo di versione come GitHub, Gitlab e Bitbucket. Questo sistema di gestione dei progetti fornisce anche diversi metodi per facilitare la migrazione da altre piattaforme software proprietarie. Taiga permette anche di effettuare chat di gruppo e private e videoconferenze, utilizzando servizi di terze parti.
Riassumendo, tramite Taiga un team ha un singolo punto che mostra lo stato di avanzamento del progetto e la suddivisione dei compiti.

Mattermost

Mattermost è una piattaforma open source che permette di comunicare in modo sicuro, collaborare ed orchestrare il lavoro dei team. È progettata per essere una chat interna

per organizzazioni ed aziende, con l'aggiunta di alcune funzionalità. Mattermost è costruita nello specifico per lo sviluppo di software e per casi d'uso ingegneristici. Tramite questo strumento le aziende possono avere il pieno controllo sui propri dati attraverso implementazioni self-hosted e private cloud. La piattaforma è completamente esendibile e supporta un ricco ecosistema di applicazioni di terze parti. Mattermost può anche essere facilmente personalizzata tramite API, framework e customizzazioni open source. Originariamente il codice di Mattermost era proprietario, dopo di che è diventato open source. La versione 1.0 è stata rilasciata nel 2015. Il progetto è gestito e sviluppato dalla società Mattermost, Inc., la quale ottiene i fondi vendendo servizi di supporto e funzionalità aggiuntive non presenti nell'edizione open source.
I tre principali strumenti presenti in Mattermost sono canali, playbook e board. I primi permettono ad un team di comunicare in real-time, sia con una chat di gruppo che con una privata tra due utenti, e faccia a faccia tramite conferenze. Tramite i canali si possono anche condividere file, immagini e link. I playbook, invece, consentono di orchestrare il lavoro dei team tramite flussi di lavoro che supportano specifici casi d'uso i quali richiedono processi affidabili e ripetibili. Questi flussi vengono eseguiti ogni volta che sorge la necessità di organizzare persone, strumenti o dati. Per quanto riguarda i partecipanti, i flussi comprendono checklist, aggiornamenti di stato e retrospettive, mentre per gli strumenti integrati configurano trigger per effettuare azioni automatiche. Infine, la board serve per il monitoraggio dei progetti tramite un insieme di attività, milestone e obiettivi, condivisi tra gli sviluppatori. La board ha una struttura simile a quella della board di Kanban e può includere immagini, documenti e link.

3.1.2 CAS Client

CAS Client è composto da due parti: la parte della dashboard, attraverso la quale si accede tramite un browser, e la parte degli IDE, utilizzati per salvare ed analizzare i dati degli sviluppatori per moitorarne la produttività.

Dashboard

La Dashboard è un componente di CAS scritto in Javascript tramite i framework Redux e React[12]. Inserita in CAS dalla versione 2.0, può essere definita come una web app che funziona da dashboard di tipo operazionale, cioè di tracciamento di processi operativi. Nel caso di CAS infatti il suo scopo è permettere ad uno sviluppatore di accedere ai suoi dati ed agli indicatori di produttività in modo aggregato.
I dati di un utente che effettua il login venogno recuperati tramite le API offerte dai diversi servizi dei quali la dashboard si occupa, per poi essere raggruppati e mostrati tramite l'utilizzo di grafici. La dashboard è concepita come hub per i servizi di CAS, perciò non c'è un unico pannello che raggruppa i dati: essi vengono divisi in sezioni, una per ogni servizio.

I dati presenti sulla dashboard comprendono anche quelli ottenuti dai logger dei diversi IDE, che permettono ad uno sviluppatore di analizzare la propria produttività e il proprio metodo di lavoro.

IDE e logger plugin

CAS Client include gli IDE open source utilizzati dagli sviluppatori per la produzione di software. Gli IDE vengono integrati con CAS tramite un plugin (*logger*) che permette di tenere traccia delle azioni degli utenti all'interno degli editor usati per scrivere codice. Eclipse, IntelliJ ed Atom sono i tre IDE sui quali è possibile installare un logger plugin ed i risultati ottenuti vengono utilizzati dal servizio Logger di CAS. Di seguito vengono mostrati i tre IDE nel dettaglio:

- **Eclipse**, un software open source distribuito sotto la licenza Eclipse Public Licence. Eclipse viene definito come un ambiente di sviluppo integrato (IDE) multilinguaggio e multipiattaforma, ideato dalla Eclipse Foundation, un'organizzazione fondata nel 2001 da diverse società quali Red Hat, IBM, Borland e SuSE, che successivamente ha incluso altre grandi aziende come HP. Eclipse è scritto in Java e viene usato principalmente per sviluppare applicazioni in tale linguaggio, ma tramite i plugin disponibili è in grado di supportare anche C/C++, XML, Javascript, PHP e molti altri. L'IDE permette anche di generare diagrammi UML, scrivere documenti in LaTeX e fornisce un metodo per progettare graficamente una GUI per un'applicazione Java. Eclipse è disponibile per diverse piattaforme, le principali sono Linux, macOS e Windows.

  Tutte queste funzionalità aggiuntive vengono integrate, come già detto, tramite dei plugin, sui quali Eclipse è incentrato. Anche la versione base è un plugin e permette di programmare in Java utilizzando diverse funzioni di aiuto quali suggerimento dei tipi di parametri dei metodi, code completion, riscrittura automatica del codice in caso di modifiche alle classi ed accesso diretto a Git.

  Il plugin per il logger può essere scaricato al seguente indirizzo: http://aminsep.disi.unibo.it/cas-project/latest/plugins/EclipseLogger_1.0.0.201901141717.jar.

- **IntelliJ IDEA**, un ambiente di sviluppo integrato per il linguaggio di programmazione Java. Sviluppato da JetBrains, è disponibile sia in un'edizione community sotto licenza Apache che in una commerciale proprietaria. La prima versione dell'IDE è stata rilasciata nel 2001 ed IntelliJ è stato tra i primi a fornire funzionalità come la navigazione avanzata del codice ed il code refactoring. L'edizione community di intelliJ è anche stata utilizzata come base di Android Studio, un IDE open source per lo sviluppo di applicazioni in Android.

Come Eclipse, IntelliJ utilizza i plugin per aggiungere funzionalità alle proprie. I plugin possono essere scaricati ed installati tramimte il sito web contenente il repository dei plugin di IntelliJ o tramite la funzione integrata di ricerca ed installazione. Ogni edizione ha i propri repository di plugin separati.

Altre funzionalità fornite dall'IDE sono quelle di assistenza alla scrittura di codice (ad esempio la code completion, il debugging ed il refactoring del codice e molte altre), l'integrazione di sistemi di controllo di versione come Git e l'accesso diretto a database quali Microsoft SQL Server, Oracle, PostgreSQL, SQLite e MySQL.

Tra i linguaggi supportati da entrambe le edizioni ci sono Java, Python e Kotlin, mentre tra quelli supportati esclusivamente dalla Ultimate Edition ci sono Javascript, PHP, HTML e SQL.

Per quanto riguarda il plugin per integrare il logger in IntelliJ, è possibile ottenere il file .jar dal repository `https://gitlab.com/fulvio1993/logger-intellij` ed installarlo come un qualsiasi altro plugin.

- **Atom**, un editor di testo ed IDE gratuito ed open source per Windows, Linux e macOS, scritto in CoffeeScript e Less. Atom è stato sviluppato da GitHub e rilasciato nel 2014. Questo IDE è un'applicazione desktop costruita usando tecnologie web e supporta diversi plugin solitamente scritti in Javascript. Atom si basa su Electron, un framework che permette la costruzione di applicazioni desktop multipiattaforma usando Chromium e Node.js.

  Questo software è altamente customizzabile, infatti presenta un init script configurabile utilizzando CoffeeScript, un foglio di stile per personalizzarne l'aspetto ed una keymap per mappare o rimappare le combinazioni di tasti ai comandi. Come la maggior parte degli editor di testo configurabili, Atom permette di installare ed utilizzare dei package di terze parti per aggiungere funzionalità. Tali package possono essere installati, gestiti e pubblicati tramite il package manager di Atom apm.

  Dei fatti degni di nota che coinvolgono Atom sono i seguenti: Electron, alla base dell'editor, è stato usato da aziende come Microsoft e Docker per alcuni dei loro prodotti e gli sviluppatori di Facebook hanno programmato Nuclide, un editor open source per uso interno derivato da una versione modificata di Atom.

  Atom-logger è un'estensione per Atom, scritta in Javascript e Less[2] ed ufficialmente pubblicata. Per utilizzarla bisogna cercarla all'interno dell'IDE come un qualsiasi altro package.

3.1.3 Keycloak e CAS

Durante questo percorso il mio scopo è stato inserire Keycloak in CAS con il ruolo di Single Sign-On, configurando SonarQube, Jenkins e GitLab per lavorare con esso. L'ambiente su cui ho lavorato è una macchina virtuale clone di aminsep, la macchina che ospita CAS per permetterne l'utilizzo da parte degli studenti. Il dominio della macchina virtuale che ho usato è **aminsep2.disi.unibo.it** ed utilizzando gli strumenti accedibili da esso tramite browser è possibile vedere i risultati che ho ottenuto.
Le varie operazioni che ho fatto sono spiegate nelle sezioni che seguono.

3.1.3.1 Scelte effettuate

Il primo passo è stato decidere quale strumento utilizzare come identity provider. La mia scelta è ricaduta su Keycloak principalmente per i seguenti motivi:

- È open source, quindi aderisce alla visione di CAS che è interamente open source anch'esso;
- Fornisce numerose funzionalità utili;
- È largamente documentato;
- Può essere facilmente utilizzato su Docker.

Un'altra scelta che ho fatto è stata quella di utilizzare OpenID Connect come protocollo di autenticazione in tutte e tre le applicazioni. L'opzione alternativa era utilizzare SAML, ma mia decisione è ricaduta su OIDC in base ai seguenti fattori:

- OpenID è più intuitivo e facile da utilizzare;
- È nettamente più recente;
- Sta prendendo sempre più piede, sostituendo SAML;
- I token che utilizza sono nel formato JSON Web Token (JWT), più leggero rispetto all'XML di SAML.

3.1.3.2 Inserire Keycloak in CAS

Per inserire Keycloak tra gli strumenti di CAS è necessario apportare a quest'ultimo alcune modifiche, riportate in dettaglio nell'appendice A. In sintesi, ho utilizzato l'immagine di Keycloak su docker per creare il suo container ed aggiungerlo a CAS come nuovo microservizio, ed ho modificato la configurazione di Nginx. Quest'ultima modifica ha il fine di porre Keycloak all'interno del reverse proxy e permettere agli utenti di effettuare richieste passando per Nginx, come succede per gli altri servizi.

Dopo avere terminato di fare queste modifiche, inserendo nel browser l'URL del dominio ed aggiungendo /keycloak sarà possbile accedere, dopo essersi loggati come amministratori, all'admin console descritta in precedenza. Inoltre, si potrà accedere ai vari endpoint citati nel capitolo su Keycloak.

3.1.3.3 Creazione di realm e client

Per utilizzare Keycloak come SSO bisogna innanzitutto creare un realm differente dal master. Il nuovo realm avrà il compito di gestire le tre applicazioni e conterrà gli utenti che verranno utilizzati da esse. Ognuna delle tre applicazioni, inoltre, deve avere un corrispettivo client nel nuovo realm di Keycloak, client che andrà quindi creato e configurato. Il modo pratico per effettuare queste azioni viene mostrato nell'appendice B.

3.1.3.4 Autoregistrazione degli utenti

Una funzionalità che ho inserito è la possibilità da parte degli utenti di registrarsi in modo autonomo, senza coinvolgere l'amministratore. Per effettuare tale operazione è sufficiente che l'utente clicchi su *New user? Register* nella schermata di login di Keycloak (3.6). In questo caso, la schermata che verrà mostrata è la seguente:

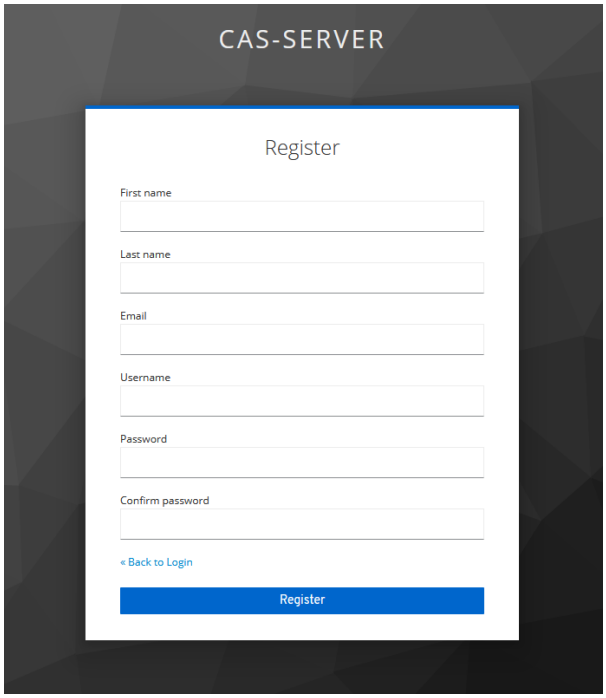

Figura 3.3: Registrazione di un utente in Keycloak

In essa bisognerà inserire i propri dati per creare un nuovo utente. I dati che vengono richiesti possono essere modificati tramite le impostazioni di Keycloak.
Come configurare un realm di Keycloak per permettere agli utenti di registrarsi è mostrato nell'appendice B.

3.1.3.5 Configurazione di SonarQube

Per poter integrare Keycloak con SonarQube ho utilizzato il plugin **OpenID Connect Authentication for SonarQube**, nel modo spiegato nell'appendice C.
Se tutto viene configurato correttamente, la schermata di login di SonarQube verrà modificata come mostrato di seguito:

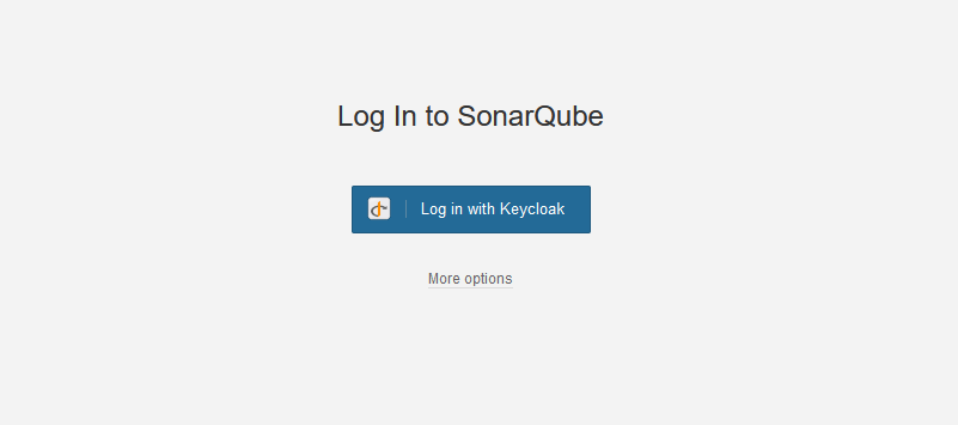

Figura 3.4: Schermata di login di SonarQube

Pemendo su *Log in with Keycloak* il browser verrà reinderizzato alla schermata di login di Keycloak (3.6), mentre premendo su *More options* si potrà accedere tramite SonarQube.

3.1.3.6 Configurazione di GitLab

Per integrare GitLab e Keycloak ho usato OmniAuth, un framework Rack che standardizza l'autenticazione multi-provider per le applicazioni web. GitLab lo utilizza per poter essere configurato in modo da permettere agli utenti di autenticarsi tramite numerosi servizi quali Twitter, GitHub ed OpenID Connect, come nel nostro caso. Le configurazioni che ho fatto vengono mostrate nell'appendice D.
Come risultato la schermata di login di GitLab viene modificata in modo simile a come succede con quella di SonarQube:

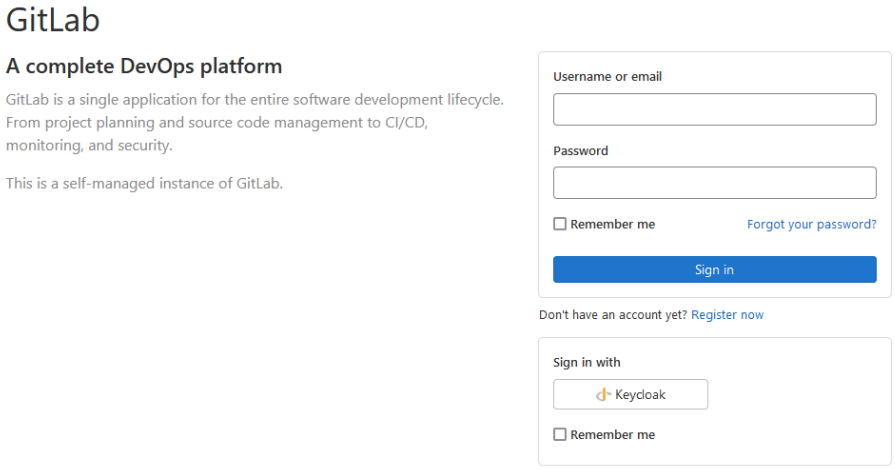

Figura 3.5: Schermata di login di GitLab

Anche in questo caso, se l'utente seleziona l'opzione Keycloak viene reinderizzato alla sua schermata di login (3.6).

3.1.3.7 Configurazione di Jenkins

Per la configurazione di Jenkins ho utilizzato il plugin: **OpenId Connect Authentication Plugin**. Nell'appendice E viene spiegato nel dettaglio come ho installato e configurato tale plugin. A differenza degli altri due servizi, la schermata di login di Jenkins viene completamente sostituita da quella di Keycloak. Quando un utente vuole accedere, quindi, gli viene mostrata la seguente pagina web:

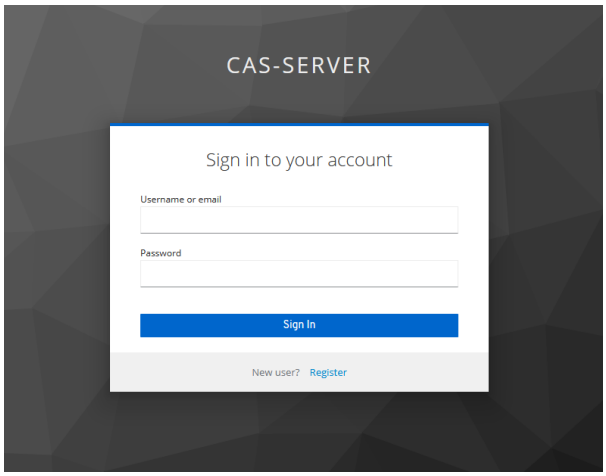

Figura 3.6: Schermata di login di Keycloak

3.1.3.8 Nuova struttura di CAS

Dopo aver apportato queste modifiche a CAS, la sua struttura cambia nel modo seguente:

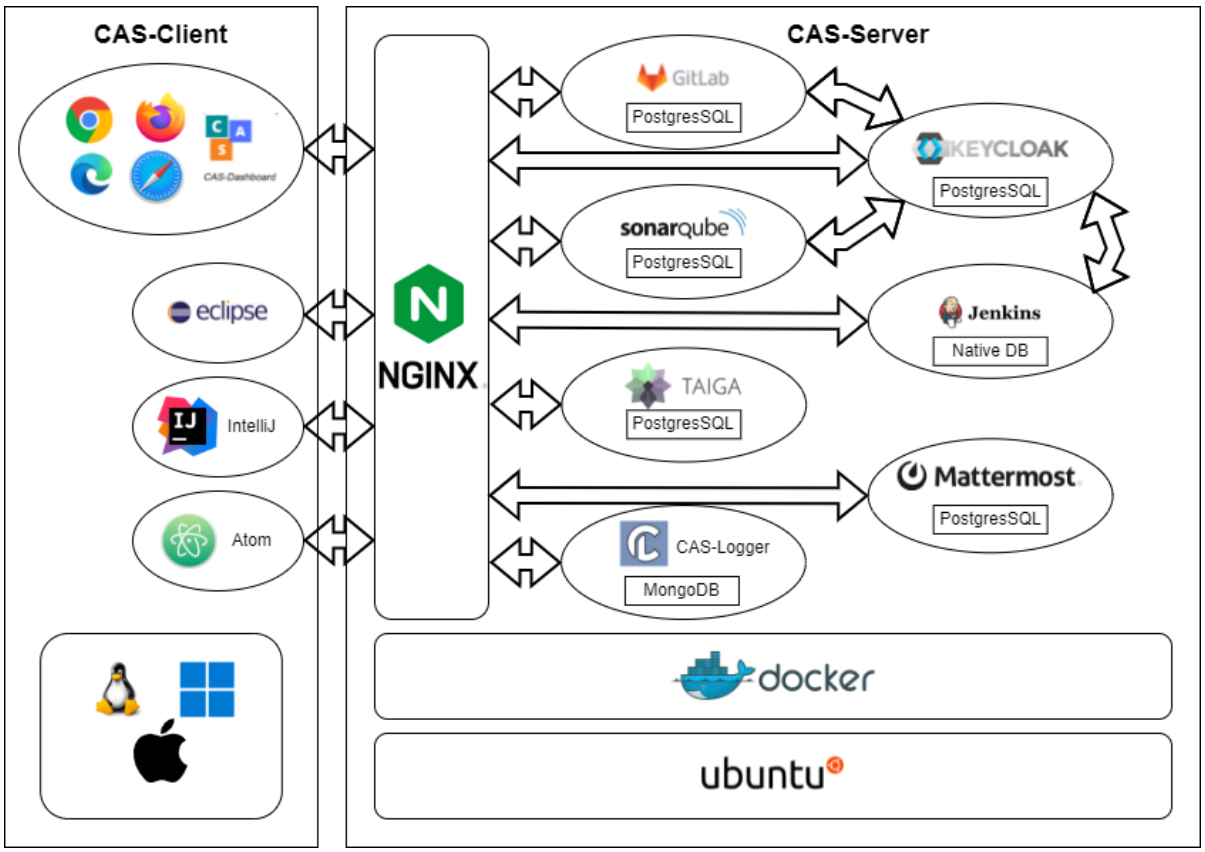

Figura 3.7: Struttura di CAS 2.0 con Keycloak

Come si vede, Keycloak viene aggiunto ai servizi esposti da CAS tramite il reverse proxy, di conseguenza vi si può accedere tramite browser. L'immagine mostra anche che Keycloak interagisce sia con Jenkins, SonarQube e GitLab che con gli utenti. L'interazione con gli utenti avviene quando essi vogliono accedere ai servizi forniti da Keycloak come l'account e l'admin console e quado devono inserire le proprie credenziali per effettuare il login. L'interazione con le applicazioni avviene durante il flusso di autenticazione. Per come ho effettuato le configurazioni, in tutti e tre i casi viene utilizzato l'Authentication Code Flow, spiegato nella sezione su OpenID Connect del capitolo 1.

CAPITOLO 4

Conclusioni

Lo scopo principale di questo lavoro è avere una visione più completa possibile della teoria dietro ad un Single Sign-On, ma anche fornire un esempio concreto di uno strumento che può svolgere tale compito e di come possa essere utilizzato in una situazione reale. Un'ulteriore finalità di questo lavoro è migliorare la User Experience nell'utilizzo dell'ambiente CAS tramite l'utilizzo di Keycloak come SSO. Di seguito voglio porre l'accento su quanto ad oggi sia diventato importante l'utilizzo di un Single Sign-On in un'applicazione.

4.1 Quanto è importante che un'applicazione utilizzi un Single Sign-On?

Dalla discussione del primo capitolo risultano chiari i vantaggi nell'utilizzo di un Single Sign-On in un ambiente di sviluppo a microservizi, quale è CAS.
Per uno sviluppatore che crea un'applicazione il vantaggio principale è non doversi preoccupare della gestione delle credenzali di utente, e di un efficace sistema di protezione. Inoltre, non dovrà neanche occuparsi della parte di login e di gestione delle operazioni che possono essere effettuate sul un profilo utente. Il motivo principale della scelta di utilizzare un SSO è però migliorare l'esperienza dell'utente.
Ormai la maggior parte delle applicazioni permettono di effettuare l'accesso tramite provider esterni, come ad esempio Facebook o Google. Se pensiamo a quante applicazioni utilizziamo ogni giorno (a lavoro, social, di svago), a mio parere risulta quasi necessario che la maggior parte di esse permetta di accedere tramite un servizio a cui siamo già registrati. Senza questa possibilità risulterebbe davvero impegnativo gestire così tante credenziali diverse.
D'altro canto, un Single Sign-On crea un single point of failure e se le credenziali vengono rubate diventa possibile accedere a tutti i servizi che le utilizzano. Tuttavia per la mag-

gior parte dei casi è sufficiente affiancare al meccanismo SSO un sistema di autenticazione forte come un dispositivo biometrico.
Concludendo, i vantaggi superano di molto gli svantaggi nella maggior parte degli scenari e l'utilizzo di un SSO è un elemento importante per la User Experience, perciò credo che la maggior parte delle applicazioni debbano fornire tale possibilità.

4.2 Problematiche riscontrate

La difficoltà principale che ho affrontato durante questo lavoro è stata la gestione e configurazione dell'ambiente CAS esistente per predisporlo all'inserimento ed utilizzo del Single Sign-On. Tra le altre cose, ho dovuto modificare delle configurazioni dei servizi, in particolare quelle riguardanti gli indirizzi IP a cui fanno riferimento. In seguito, ho anche lavorato con le impostazioni del DNS, con le configurazioni di Nginx e con Certbot per la creazione di certificati.
Questa parte è quella che ha occupato la maggiore porzione del tempo, infatti ho dovuto risolvere diverse problematiche che non avevo mai incontrato.

4.3 Lavori futuri

Il lavoro che ho effettuato costituisce una buona base per possibili ampliamenti futuri, avendo inserito uno strumento open source come Keycloak a mio parere molto potente. Di seguito elenco alcune delle espansioni che potrebbero essere fatte:

4.3.1 Usufruire di ulteriori funzionalità di Keycloak

Keycloak è uno strumento che fornisce diverse funzionalità utili. Nel nostro caso è stato usato come Single Sign-On, ma si potrebbe pensare anche ad esempio di configurare un'autenticazione a più fattori o permettere ad uno studente di utilizzare le proprie credenziali istituzionali per effettuare il login. Per quest'ultima parte credo si potrebbe usare l'identity brokering, nel caso in cui l'ateneo fornisca un identity provider, altrimenti si potrebbe valutare l'utilizzo dell'user storage federation.

4.3.2 Aggiunta del Single Sign-0n ad altri servizi CAS

Al momento sono rimasti fuori del SSO che ho realizzato per CAS i seguenti servizi: Taiga, Mattermost e Logger Dashboard.
Nel mio lavoro non ho potuto inserire Taiga poiché ho incontrato delle problematiche che non sono riuscita a risolvere, mentre per Mattermost non ho trovato sufficiente materiale o documentazione. Un ulteriore miglioramento che si potrebbe apportare sarebbe, quindi,

la configurazione di ulteriori strumenti di CAS diversi da Jenkins, SonarQube e GitLab per permettergli di delegare l'autenticazione a Keycloak. Oltre a Taiga e Mattermost, si potrebbe inserire la dashboard, per la quale sarebbe necessaria la modifica del codice. Penso che utilizzando la base risultante da ciò che ho fatto sia possibile concentrarsi maggiormente sullo studio degli strumenti rimanenti per integrarli.

Bibliografia

[1] The Agile Alliance. Manifesto for agile software development. http://agilemanifesto.org.

[2] Carlos Caramaschi, Salvatore Perri, and Stefano Propato. atom-logger package page. https://atom.io/packages/atom-logger.

[3] Debdut Chakraborty. Using docker to set up nginx reverse proxy with auto ssl generation. https://linuxhandbook.com/nginx-reverse-proxy-docker/.

[4] Paolo Ciancarini, Marcello Missiroli, Francesco Poggi, and Daniel Russo. An open source environment for an agile development model. In *Proc. 16th Int. Conf. on Open Source Systems (OSS)*, volume 582 of *IFIP Advances in Information and Communication Technology*, pages 148–162, Innopolis, Russia, 2020. Springer.

[5] Docker. Docker run reference. https://docs.docker.com/engine/reference/run/.

[6] GitBook. Keycloak documentation. https://wjw465150.gitbooks.io/keycloak-documentation/content/index.html.

[7] InnopolisUniversity. innometrics-backend github repository. https://github.com/InnopolisUniversity/innometrics-backend. (Consultato in data: 01/03/2021).

[8] InnopolisUniversity and Stefano Propato. logger-backend github repository. https://github.com/elPeroN/logger-backend.

[9] Keycloak. Documentation. https://www.keycloak.org/documentation.

[10] Keycloak. Documentation. https://www.keycloak.org/docs/latest/server_admin/#_identity_broker.

[11] Keycloak. Getting started guide. https://www.keycloak.org/docs/latest/getting_started/index.html.

[12] Stefano Propato and Salvatore Perri. Cas-dashboard github repository. https://github.com/elPeroN/CAS-dashboard.

[13] Santosh Shinde. Introduction to building an effective identity and access management architecture with keycloak. https://javascript.plainenglish.io/introduction-to-building-an-effective-identity-and-access-management-architecture

[14] Stian Thorgersen and Pedro Igor Silva. *Keycloak - Identity and Access Management for Modern Applications*. Packt Publishing Ltd, 05 2021.

[15] Yvonne Wilson and Abhishek Hingnikar. *Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0*. APress, 2019.

APPENDICE A

Aggiunta di Keycloak in CAS

A.1 Aggiornamento di Nginx

Innanzitutto ho modificato il file di configurazione di Nginx per aggiungere Keycloak tra i servizi a cui vengono passate le richieste. Per accedere al file di configurazione ho utilizzato i seguenti comandi:

```
$ sudo docker exec -it nginx bash
```

Listing A.1: Comando per accedere alla bash di Nginx

```
$ vi etc/nginx/conf.d/default.conf
```

Listing A.2: Comando per modificare la configurazione di Nginx

Il primo permette di accedere alla bash di Nginx, mentre il secondo utilizza un editor per modificare il file indicato.
All'interno del file default.conf, poi, ho aggiunto le righe di codice necessarie per inserire Keycloak tra i servizi ai quali si può effettuare una richiesta. Il codice da inserire è quello presentato d seguito.

```
upstream keycloak {
    server keycloak:8080;
}
```

Listing A.3: Inserimento upstream di Keycloak in Nginx

Il codice mostrato sopra va inserito nella parte iniziale del file, aggiungendolo a quello degli altri upstream, mentre quello che segue va inserito nella parte del server.

```
location ^~ /keycloak {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://keycloak/;
    }

    location /auth {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://keycloak/auth;

        proxy_buffer_size 128k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;
    }
```

Listing A.4: Configurazione degli URL di Keycloak in Nginx

In questo caso, la parte che modifica la dimensione del buffer è necessaria per permettere agli utenti di registrarsi a Keycloak senza errori.

A.2 Aggiunta di Keycloak dockerizzato

Per aggiungere Keycloak a CAS ho utilizzato l'immagine di Keycloak su docker per mantenere l'architettura a microservizi di questo ambiente. È possibile eseguire Keycloak su un container docker in due diversi modi; il primo è utilizzare il comando **docker run**[5] nel modo seguente:

```
docker run -d --name keycloak --network cas-server_default -e
    KEYCLOAK_USER=admin -e KEYCLOAK_PASSWORD=admin -e DB_VENDOR=h2 -e
    PROXY_ADDRESS_FORWARDING=true jboss/keycloak
```

Listing A.5: Creare il container di Keycloak con docker run

In questo caso l'immagine utilizzata è jboss/keycloak ed il nome del container è Keycloak. Inoltre viene creato un utente iniziale con il ruolo di amministratore le cui credenziali sono utente *admin* e password *admin*. Il database utilizzato è h2 che in questo caso è incapsulato in Keycloak.
Il secondo modo per farlo è modificare il file **docker-compose.yml** all'interno di CAS-Server, aggiungendo le seguenti linee:

```
#KEYCLOAK
  postgres:
      container_name: keycloak-db
      image: postgres
      volumes:
        - postgres_data:/var/lib/postgresql/data
      environment:
        POSTGRES_DB: keycloak
        POSTGRES_USER: keycloak
        POSTGRES_PASSWORD: password

  keycloak:
    container_name: keycloak
    image: 'jboss/keycloak'
    environment:
      - DB_VENDOR=POSTGRES
      - DB_ADDR=keycloak-db
      - DB_DATABASE=keycloak
      - DB_USER=keycloak
      - DB_SCHEMA=public
      - DB_PASSWORD=password
      - KEYCLOAK_USER=admin
      - KEYCLOAK_PASSWORD=admin
      - PROXY_ADDRESS_FORWARDING=true
    volumes:
      - realm-config:/opt/jboss/keycloak/realm-config
      - keycloak-db:/opt/jboss/keycloak/standalone/data
    restart: always
    networks:
      - default

volumes:
  #keycloak
  realm-config:
  keycloak-db:
  postgres_data:
```

Listing A.6: Creare il container di Keycloak con docker compose

Come si può vedere, le configurazioni del container sono simili a quelle utilizzate nel **docker run**, con la differenza che in questo caso il database utilizzato è postgres ed è esterno. Dopo aver modificato il file, è sufficiente eseguire il comando

```
docker compose up -d
```

Listing A.7: Docker compose

A.3 Repository GitHub

Una versione di CAS in cui Keycloak sia già integrato è anche disponibile presso il repository GitHub https://github.com/f-guerzoni/CAS-Server, un fork del repository di CAS-Server originale: https://github.com/elPeroN/CAS-Server.
A questo punto, è sufficiente eseguire il seguente comando per clonare il repository:

```
$ git clone https://github.com/f-guerzoni/CAS-Server.git
```

Listing A.8: Comando per clonare il repository di CAS-Server con Keycloak

Per poi avviare l'installazione tramite uno script, eseguendo il comando:

```
$ ./cas.sh -i
```

Listing A.9: Comando per installare CAS-Server con Keycloak

APPENDICE B

Configurazione di Keycloak

Per utilizzare Keycloak come Single Sign-On su Jenkins, SonarQube e GitLab ho creato e modificato due elementi di Keycloak: realm e client.

B.1 Creazione di un realm

Il primo passo è creare un realm su Keycloak all'interno del Master Realm presente di default. Per effettuare tale operazione bisogna passare il mouse sul menu a discesa nell'angolo in alto a sinistra intitolato *Master*. Questo menu elenca tutti i realm creati e l'ultima voce è sempre *Add realm*. Cliccando tale opzione, viene mostrata la schermata seguente:

Figura B.1: Creazione di un nuovo realm in Keycloak

All'interno del campo *Name* va inserito il nome del nuovo realm, nel mio caso lo ho chiamato CAS-Server. In questo passaggio è bene tenere conto del fatto che il nome del realm è case-sensitive.

B.2 Abilitare auto-registrazione

Una volta creato il realm bisogna selezionarlo dal menu a tendina citato nella sezione precedente ed iniziare a configurarlo. Le operazioni principali che ho fatto sono creare dei client, come spiegato successivamente, ed abilitare l'auto-registrazione degli utenti.

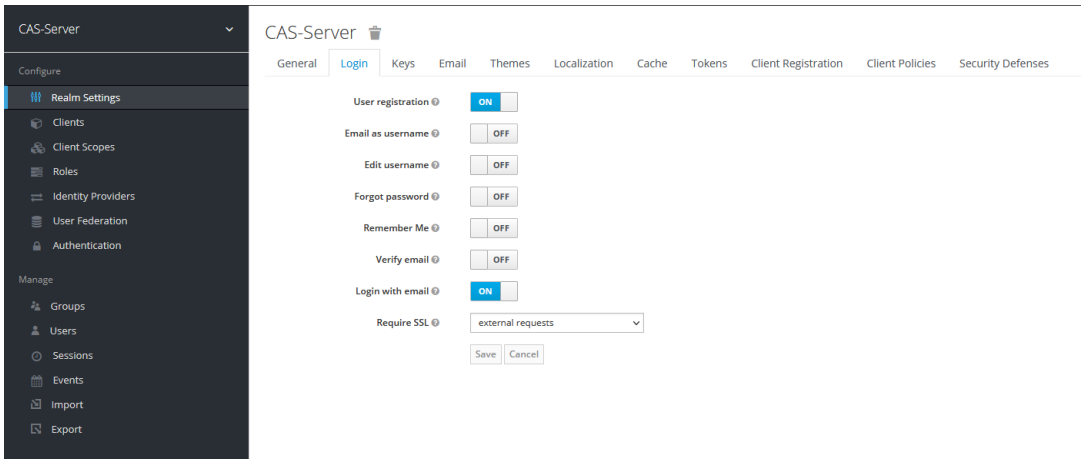

Figura B.2: Abilitare la registrazione in Keycloak

Come mostrato, è sufficiente andare su *Realm Settings* > *Login* e impostare *User registration* ad on. Da questa schermata è anche possibile abilitare altre funzionalità utili.

B.3 Creazione di un client

Per permettere ad un certo elemento di delegare l'autenticazione a Keycloak è necessario creare un client corrispondente a tale strumento su Keycloak. Il concetto di client è stato discusso nel capitolo 2. Per creare un client bisogna andare sulla voce *Clients* del menu del realm e premere su *Create* in alto a destra. La schermata mostrata è la seguente:

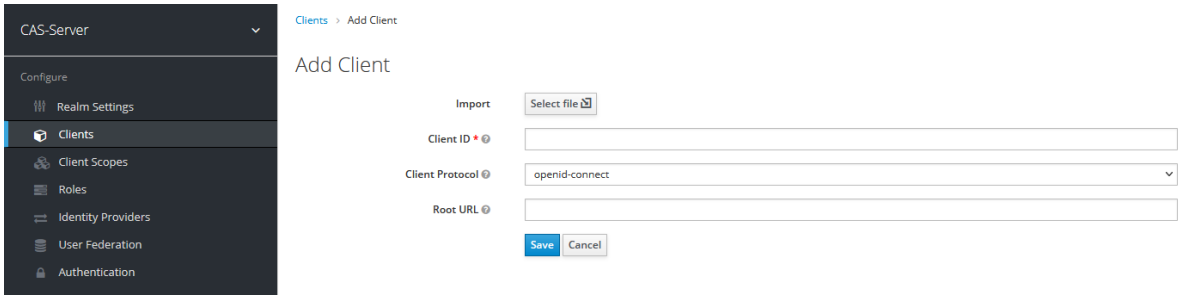

Figura B.3: Creazione di un nuovo client in Keycloak

I parametri da inserire sono l'ID del client e il suo URL, inoltre si può selezionare il protocollo usato; in tutti e tre i casi ho selezionato *openid-connect*.
Dopo aver compilato i campi richiesti si arriva alla schermata delle impostazioni del client. Per tutti e tre i servizi ho cambiato l'impostazione *Access Type* in confidential, selezionandolo dal menu dropdown. In questo modo il client manterrà un client secret, il quale può essere reperito accedendo a *Credentials*, come mostrato in figura:

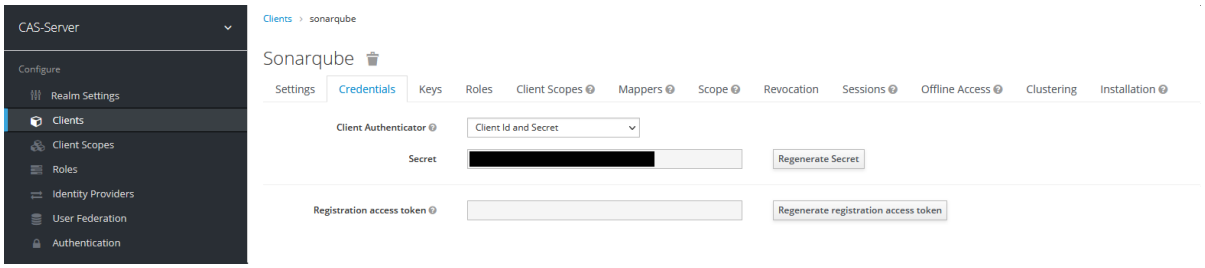

Figura B.4: Client secret di un client Keycloak

APPENDICE C

Configurazione di SonarQube

C.1 Creazione del client

Innanzitutto, bisogna creare un client su Keycloak nel modo spiegato nell'appendice B. Come ID ho insertito *sonarqube* e come root URL https://{dominio}/sonarqube. Come già detto, ho anche modificato l'access type, impostandolo su confidential.

C.2 Installazione plugin

Per installare il plugin ho effettuato l'accesso a SonarQube come amministratore, dopo di che sono andata su *Administration* > *Marketplace* e l'ho cercato tra l'insieme di plugin disponibili.

C.3 Configurazione plugin

Prima di tutto bisogna controllare che l'URL di SonarQube sia corretto. Tale URL può essere controllato e modificato alla sezione *Administration* > *Configuration* > *General Settings* > *General* > *Server Base URL*.
La configurazione di OpenID che ho fatto è la seguente:

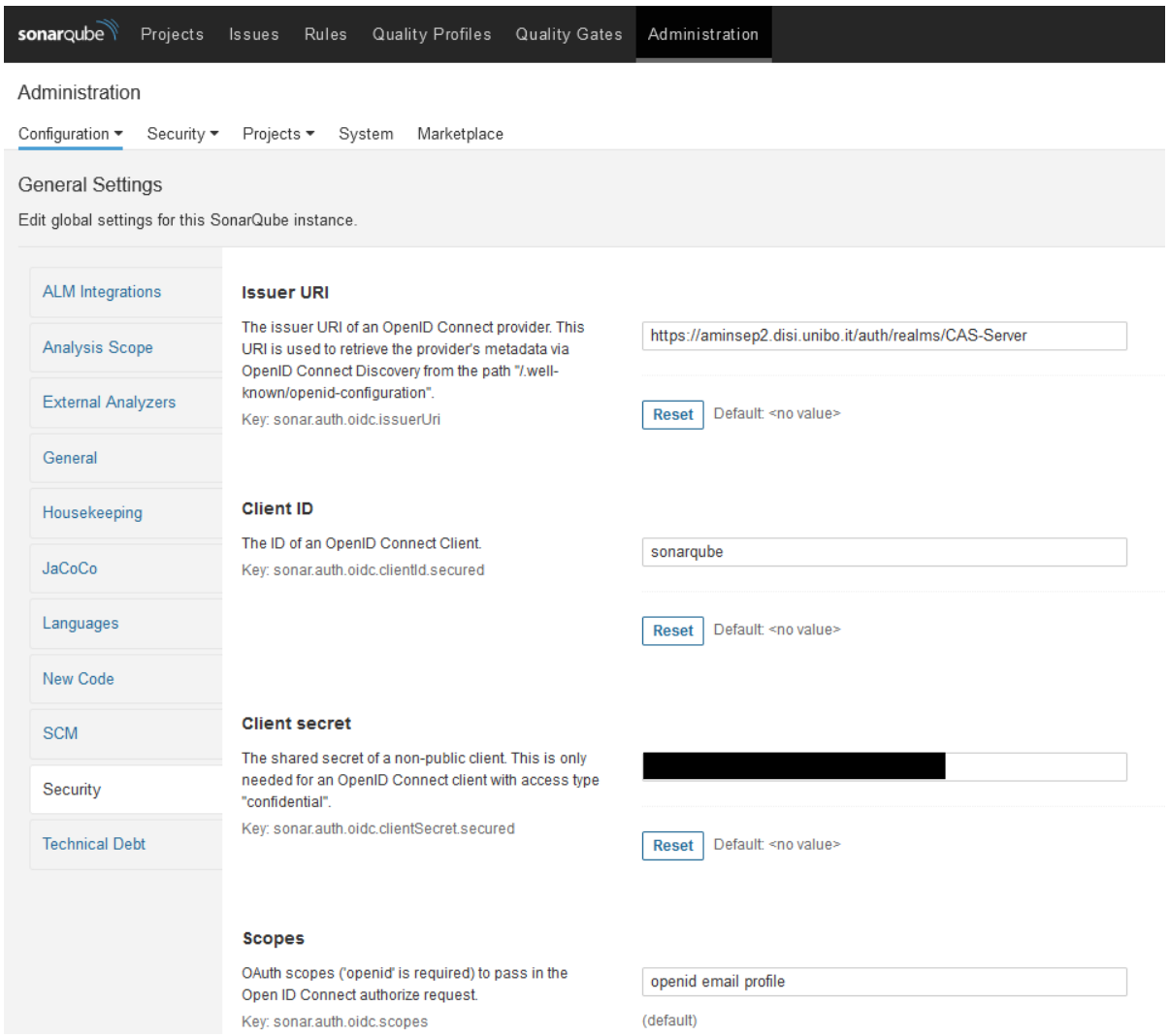

Figura C.1: Configurazione di OpenID su SonarQube

Come si vede, anche per il plugin le impostazioni da confgurare sono in *General Settings*, ma nella sottosezione *Security*. Per poter utilizzare OpenID bisogna configurare correttamente i parametri come mostrato in figura, inserendo nell'*Issuer URI* il dominio corretto ed il nome corretto del realm, oltre ovviamente ad inserire nel *Client secret* il segreto ottenunto nel modo precedentemente spiegato.

APPENDICE D

Configurazione di GitLab

D.1 Creazione del client

Il primo passo è creare un nuovo client nel realm, che nel nostro caso è *CAS-Server*, come mostrato nell'appendice B. Ho inserito *gitlab* come client ID e nell'URL ho inserito https://{dominio}/gitlab. Ho impostato il client come confidential e ho ottenuto il client secret, usato in seguito.

D.2 Configurazione Omnibus GitLab

Per utilizzare OmniAuth ho modificato la parte del file **docker-compose.yml** riguardante le configurazioni Omnibus di GitLab come mostrato di seguito.

```
#GITLAB
  gitlab2021:
    container_name: gitlab2021
    image: 'gitlab/gitlab-ee:latest'
    restart: unless-stopped
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'https://{dominio}/gitlab'
        gitlab_rails['gitlab_shell_ssh_port'] = 2224
        gitlab_rails['omniauth_allow_single_sign_on'] = ['
    openid_connect']
        gitlab_rails['omniauth_block_auto_created_users'] = false
        gitlab_rails['omniauth_sync_profile_from_provider'] = ['
    openid_connect']
        gitlab_rails['omniauth_sync_profile_attributes'] = ['name', '
    email']
        gitlab_rails['omniauth_providers'] = [
        {
```

```
16          'name':'openid_connect',
17          'label':'Keycloak',
18          'args':{
19            'name':'openid_connect',
20            'scope':['openid', 'profile', 'email'],
21            'response_type':'code',
22            'issuer':'https://{dominio}/auth/realms/{realm-name}',
23            'client_auth_method':'query',
24            'discovery':true,
25            'uid_field':'preferred_username',
26            'client_options':{
27              'identifier':'gitlab',
28              'secret':'******************************',
29              'redirect_uri':'https://{dominio}/gitlab/users/auth/
    openid_connect/callback'
30            }
31          }
32        }
33        ]
34        letsencrypt['enable'] = false
35        nginx['listen_port'] = 80
36        nginx['listen_https'] = false
37      volumes:
38        - gitlab_config:/etc/gitlab
39        - gitlab_logs:/var/log/gitlab
40        - /opt/gitlab2021:/var/opt/gitlab
41      restart: unless-stopped
42      networks:
43        - default
```

Listing D.1: Modificare GitLab per integrarlo con Keycloak

Quella mostrata è, dunque, la parte di GitLab del compose file modificata. Occorre ricordarsi di sostituire {dominio} con il dominio che si sta usando, {realm-name} col nome del realm ed impostare il corretto client secret, ottenuto come spiegato in precedenza.

D.3 Logout

Se si vuole che effettuando il logout da GitLab anche la sessione su Keycloak venga interrotta bisogna inserire nell'impostazione *After sign-out path* l'endpoint di logout di Keycloak, spiegato nella sezione che parla delle feature di Keycloak del capitolo 2 (Supporto per OpenID Connect).

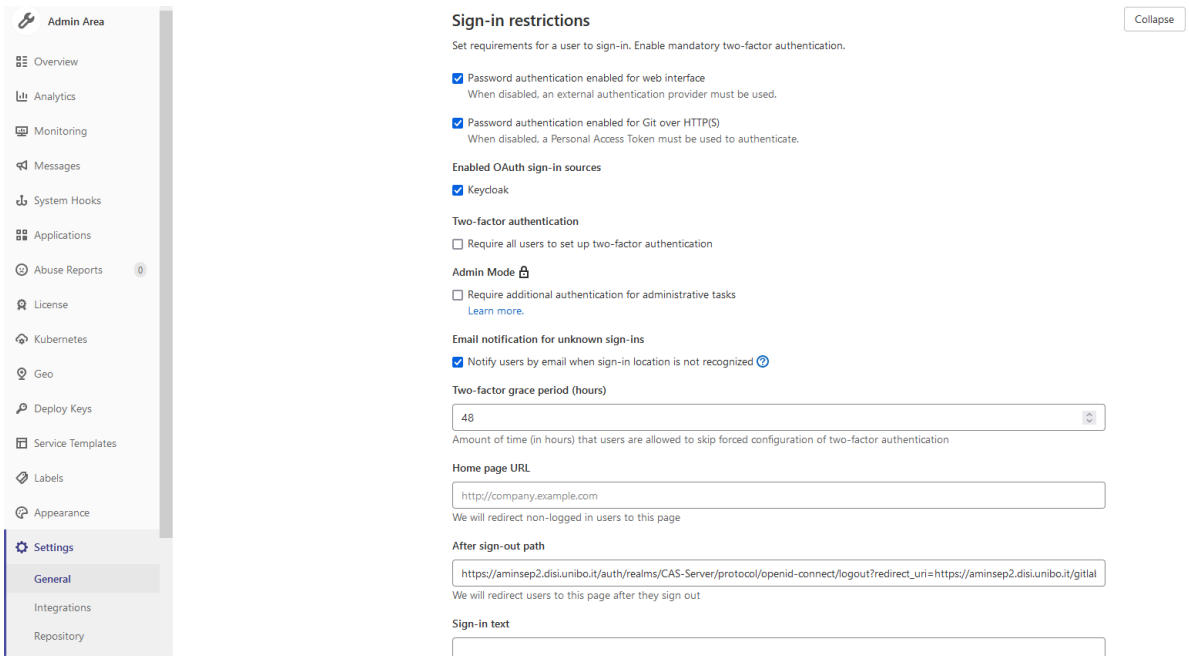

Figura D.2: Modificare l'after sign-out path di GitLab

Come mostra la figura, l'*After sign-out path* è accedibile in *Admin Area* > *Settings* > *General* > *Sign-in restrictions*.

APPENDICE E

Configurazione di Jenkins

E.1 Creazione del client

Ho creato un client in Keycloak (appendice B) con ID jenkins e root URL https://{dominio}/jenkins, ed ho impostato il client come confidential.

E.2 Installazione plugin

Il plugin è installabile come un qualsiasi altro plugin di Jenkins nella sezione *Gestisci Jenkins > Gestisci componenti aggiuntivi > Disponibili*.

E.3 Configurazione plugin

Infine bisogna andare in *Gestisci Jenkins > Configura sicurezza globale* e nella parte *Area di sicurezza* abilitare il login con Openid Connect e compilare i campi per la configurazione. Io li ho compilati nel modo seguente:

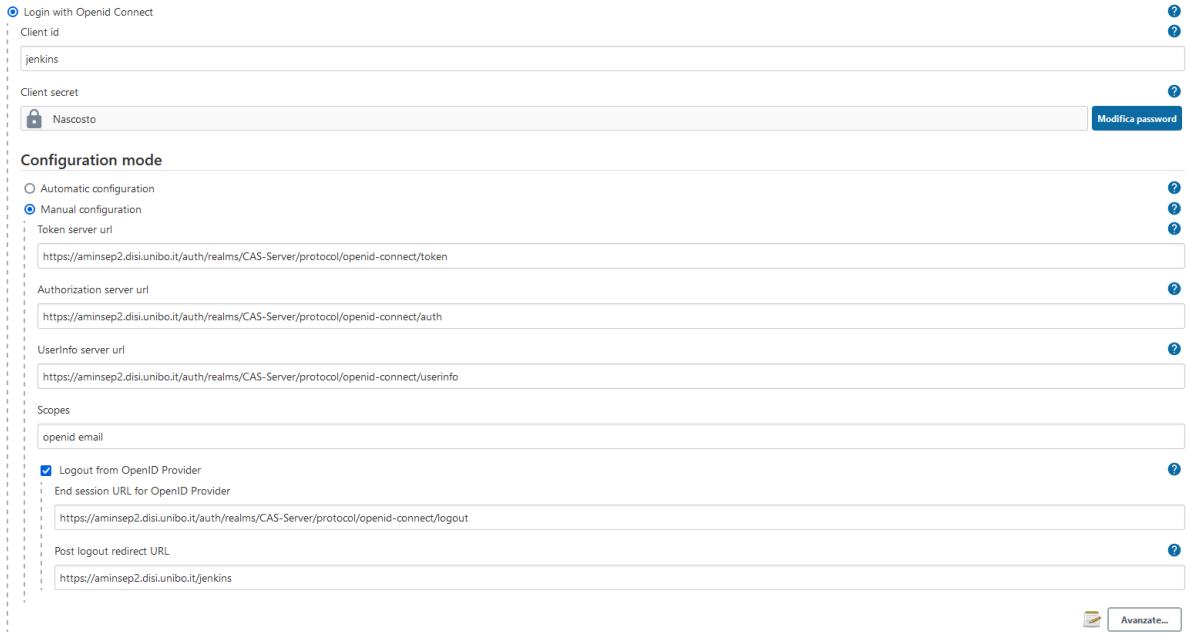

Figura E.1: Configurazione di OpenID su Jenkins

Innanzitutto ho inserito il client ID ed il client secret di Jenkins, dopo di che ho selezionato la configurazione manuale, nella quale vanno inseriti gli endpoint richiesti. Gli endpoint da utilizzare sono ricavabili dall'URL well-known di Keycloak e vanno inseriti col giusto dominio, che nel mio caso è *aminsep2.disi.unibo.it*, e col giusto nome del realm (*CAS-Server*). Bisogna anche accedere alla parte di impostazioni avanzate e modificare i parametri come mostrato in figura:

Figura E.2: Configurazione avanzata di OpenID su Jenkins

In tal modo viene effettuato correttamente il mapping tra i campi dell'ID token e gli attributi dell'utente su Jenkins.
Infine, bisogna assicurarsi che l'*URL di Jenkins* sia corretto. Per controllare ed eventualmente modificarlo bisogna andare su *Gestisci Jenkins* > *Configura sistema* > *Percorso di Jenkins*.

Ringraziamenti

Innanzitutto ringrazio il mio relatore, il Prof. Ciancarini, per la grande disponibilità dimostrata e per avermi giuidata in questo percorso. Grazie anche al Prof. Missiroli, mio correlatore, per la gentilezza che ha sempre mostrato nei miei confronti.

Un ringraziamento speciale va alla mia famiglia per avermi sempre sostenuta e avermi permesso di intraprendere questo percorso. Ringrazio in particolare mia madre e mia sorella per essere state la mia roccia, aver creduto in me ed avermi aiutata a superare i momenti difficili, senza di voi non sarei arrivata fino a qui.

Ringrazio i miei compagni Paolo, Gabriele e Nicholas per aver condiviso con me questo percorso ed essere stati al mio fianco sia nei momenti belli di risate e spenzieratezza, che in quelli brutti di ansia e difficoltà.

Grazie ai miei amici, in particolare ad Enrico per essere stato sempre presente e per avermi aiutata nel momento del bisogno.

Infine, ringrazio il mio fidanzato Marco per avermi supportata e sopportata durante questi anni, avermi tranquillizzata ed incoraggiata e non aver mai dubitato delle mie capacità né del fatto che sarei riuscita ad arrivare a questo punto.