



Università
di Catania

UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Lorenzo Didomenico

UserGate: un LDAP Translucent Proxy per la gestione
degli utenti UNICT

RELAZIONE PROGETTO FINALE

Relatore: Prof. Mario Di Raimondo

Anno Accademico 2023 - 2024

*Alla mia famiglia, ai miei amici e al mio paese
A Gabriel che aveva il mio stesso sogno
A Nico che aspetto ancora in campo*

Indice

1	Introduzione	5
2	Standard per la gestione di utenti	7
2.1	Servizi di Directory	7
2.1.1	Caratteristiche di un Servizio di Directory	7
2.1.2	Elementi di un Servizio di Directory	8
2.1.3	Entries di una Directory	8
2.1.4	Accesso ad un Servizio di Directory	8
2.2	Active Directory	9
2.2.1	Caratteristiche di Active Directory	9
2.2.2	Scopo di Active Directory	10
2.2.3	Active Directory Domain Services	11
2.2.4	Struttura gerarchica di Active Directory	11
2.2.5	Active Directory Replication	13
2.2.6	Active Directory vs LDAP	14
2.3	Protocollo LDAP	14
2.3.1	Storia del protocollo LDAP	14
2.3.2	Caratteristiche del protocollo LDAP	15
2.3.3	Struttura di una directory LDAP	16
2.3.4	Componenti di LDAP	17
2.3.4.1	DNs e RDNs	18
2.3.4.2	Attributi LDAP	18
2.3.4.3	Classi di oggetti	18
2.3.4.4	Object Identifier(OIDs)	19
2.3.4.5	Operazioni di ricerca LDAP	19
2.3.4.6	Filtri di ricerca LDAP	20
2.3.4.7	Modifiche e tipi di modifiche	21
2.3.5	Processo di autenticazione LDAP	22
2.3.6	Differenza tra LDAPv2 e LDAPv3	23

<i>INDICE</i>	3
---------------	---

3 Tecnologie Usate	24
3.1 OpenLDAP	24
3.1.0.1 Caratteristiche di OpenLDAP	24
3.1.0.2 Componenti di OpenLDAP	25
3.1.0.3 Installazione e configurazione di OpenLDAP	25
3.1.0.4 Lista degli strumenti di OpenLDAP	26
3.1.0.5 Gli OpenLDAP overlays	27
3.2 Tecnologie per lo sviluppo	29
3.2.0.1 HTML	29
3.2.0.2 CSS	30
3.2.0.3 Javascript	30
3.3 Tecnologie di supporto	31
3.3.1 Node.js e Npm	31
3.3.1.1 Node.js	31
3.3.1.2 Npm	31
3.3.2 Moduli Javascript utilizzati	32
3.3.3 EJS (Embedded Javascript)	33
3.3.4 Container e Docker	33
3.3.4.1 Caratteristiche dei container	33
3.3.4.2 Docker	34
3.3.4.3 Docker container networking	36
4 Realizzazione del progetto	38
4.1 Configurazione del container Proxy OpenLDAP	38
4.2 Configurazione del Container App	42
4.3 Configurazione della rete di container	44
4.4 Configurazione del Proxy OpenLDAP	46
4.5 Translucent Proxy Admin: struttura e implementazione	51
4.5.1 Struttura dell'applicazione	52
4.5.1.1 Design pattern model-view-controller	52
4.5.1.2 Architettura three-tier	54
4.5.1.3 Il concetto di API RESTful	54
4.5.2 Codice dell'applicazione	56
4.5.2.1 Main dell'applicazione	56
4.5.2.2 Routes dell'applicazione	56
4.5.2.3 Controller dell'applicazione	58
4.5.3 Views dell'applicazione	61
4.6 Requisiti dell'applicazione	63
4.6.1 Funzionalità previste dall'OpenLDAP Translucent Proxy	63
4.6.2 Funzionalità previste dall'applicazione web	64
4.7 Esempio utilizzo dell'applicazione	65

<i>INDICE</i>	4
Conclusione	70
Bibliografia	72

Capitolo 1

Introduzione

La gestione efficiente di utenti e risorse è fondamentale al giorno d'oggi per il corretto funzionamento ed il successo di un'azienda, grande o piccola che sia. In qualsiasi momento un amministratore vorrebbe poter accedere ad informazioni utili sulle entità di un'azienda, mentre un utente vorrebbe poter utilizzare le risorse messe a disposizione autenticandosi in maniera veloce. Per far questo gran parte delle compagnie odierne utilizzano i servizi di directory. Un servizio di directory è un database che memorizza informazioni su utenti e risorse di una rete e le gestisce in maniera organizzata. Uno tra i più famosi servizi di directory è sviluppato da Microsoft ed è intitolato Microsoft Active Directory. L'Università di Catania , come tutti gli enti pubblici, ha negli anni trovato il bisogno di gestire le informazioni riguardo ad un numero elevatissimo di utenti e di dare ad essi autorizzazioni ed accesso a risorse messe a disposizione dalla rete universitaria. Proprio per questo anche UNICT basa questa gestione sul servizio di directory Microsoft Active Directory, che permette l'archiviazione di informazioni su utenti e oggetti nella rete. Uno degli aspetti del servizio di directory che l'amministrazione dell'Università di Catania sfrutta è la possibilità di suddividere le utenze in gruppi, ovvero di poter creare particolari raggruppamenti in base alle esigenze e gestire i controlli di accesso e le autorizzazioni a particolari servizi sulla base della membership degli utenti a questi, in modo da garantire controlli di sicurezza più granulari.

Questo aspetto è gestito in maniera globale dall'Università di Catania, che quindi gestisce un insieme elevato di utenze. Nel caso più specifico, alcuni degli esempi di gruppi nei quale sono divisi gli studenti dell'Università di Catania sono ad esempio gruppi dipartimentali, oppure gruppi Erasmus, o gruppi di Studenti abilitati all'accesso VPN etc...

Potrebbe tornare utile, per ogni dipartimento, avere un sistema locale di gestione di utenti e gruppi, permettendo quindi, ad esempio, la creazione di

entità o membership locali in maniera veloce e sicura, sulla quale poi verrano garantiti particolari accessi a risorse del dipartimento. Possiamo fare un esempio banale: supponiamo che nel nostro dipartimento nasca l'esigenza di creare un gruppo "Test" ed aggiungervi solo particolari utenti del dipartimento, in modo da dar loro autorizzazioni particolari. Questo dovrebbe essere gestito in maniera globale a livello d'ateneo e potrebbe essere poco scalabile.

L'intento di questa tesi è la presentazione di un progetto, basato su OpenLDAP, che ha come obiettivo la creazione di un servizio di gestione utenze trasparente al di sopra dell'Active Directory d'ateneo. Lo scopo principale è mettere a disposizione di ogni dipartimento una piattaforma che, se da un lato presenta le stesse informazioni memorizzate dall'Active Directory originario, dall'altro permetta una gestione interna di gruppi ed utenze locali. Il progetto comprende anche l'implementazione di un'applicazione web, chiamata *Translucent Proxy Admin*, che ha come obiettivo quello di presentare un'interfaccia web user-friendly per facilitare la gestione dell'OpenLDAP Server agli amministratori.

Nel corso del secondo capitolo si parlerà nel dettaglio degli strumenti e dei protocolli utilizzati al giorno d'oggi per la gestione delle utenze e delle risorse all'interno di un'azienda, della loro storia e delle loro caratteristiche principali.

Nel terzo capitolo si parlerà delle tecnologie utilizzate per la realizzazione del progetto, descrivendone le loro caratteristiche principali.

Nel quarto capitolo si descrivono i passaggi fatti nella realizzazione del progetto sopra descritto, sia nella parte configurativa dell'OpenLDAP server sia nella parte implementativa dell'applicazione web.

Capitolo 2

Standard per la gestione di utenti

2.1 Servizi di Directory

2.1.1 Caratteristiche di un Servizio di Directory

Un **Servizio di Directory** è un programma o un insieme di programmi che provvedono ad organizzare, gestire e memorizzare informazioni e risorse centralizzate e condivise all'interno di reti di computer, rese disponibili agli utenti, fornendo anche un controllo degli accessi sull'utilizzo delle stesse, utile al lavoro dell'amministratore di sistema. Le informazioni spesso memorizzate dai **Servizi di Directory** sono informazioni come password, username, mail, membership, informazioni su device etc... Ad esempio, quando un utente vuole accedere ad un'applicazione, questa applicazione si rivolgerà ad un Servizio di Directory per assicurarsi che l'utente abbia i privilegi adatti per usarla. I servizi di directory forniscono dunque uno strato di astrazione intermedio tra le risorse da una parte e gli utenti dall'altra. Con **directory** indichiamo un database organizzato secondo una modalità gerarchica. Questa è la principale differenza dai tradizionali database relazionali. Una **directory** riceve quasi esclusivamente accessi in lettura, mentre le operazioni di scrittura sono limitate esclusivamente agli amministratori di sistema. Ne segue quindi che le directory non sono adatte ad immagazzinare informazioni aggiornate di frequente.

2.1.2 Elementi di un Servizio di Directory

Un Servizio di Directory è di solito implementato via software ed è distribuito su più server per garantire scalabilità, efficienza ed affidabilità. I Servizi di directory forniscono:

- Uno schema che descrive diversi oggetti della directory (ad esempio account utente, server, stampanti...) ed i loro attributi (ad esempio nome, indirizzo, numero di cellulare...)
- Un database universale o un catalogo che contiene informazioni dettagliate su ogni oggetto nella directory
- Indici e metodi di query utili ad utenti, amministratori e applicazioni per ottenere informazioni dalla directory
- Funzioni di replica per distribuire informazioni della directory su più server sincronizzati

2.1.3 Entries di una Directory

Una directory è una collezione di entries, che consistono ognuna in una o più attributi. Ogni attributo ha una o più valori ed un tipo che determina il tipo di informazione che il valore può ospitare e come questi valori si devono comportare durante determinate operazioni sulla directory. Questi tipi di attributi e le classi di oggetti sono descritti nell'RFC 4510 (Request for comments, documento che contiene informazioni e specifiche su un determinato protocollo). Le entries sono adattate gerarchicamente in un albero che è strutturato geograficamente e in base alle esigenze organizzative. Le entries globali, come le nazioni o le regioni, stanno nella radice dell'albero, seguite dallo stato o dalle nazioni, poi dalle organizzazioni di persone, di device etc... Una entry di una directory è rappresentata da un nome, da un *Relative Distinguished Name* e da un *Distinguished Name*. Il DN identifica in maniera univoca ogni entry a livello globale, ed è ottenuto concatenando l'RDN di una entry con il DN dell'entry antenata.

2.1.4 Accesso ad un Servizio di Directory

Le directory usano un protocollo di accesso semplificato ed ottimizzato, il **protocollo LDAP**. Esso è il protocollo standard per l'accesso ai **servizi di directory**. Un'applicazione che vuole leggere o scrivere informazioni in una directory non vi accede direttamente, ma invoca una funzione o un'API che genera l'invio di un messaggio ad un altro processo. Questo secondo processo

accede alle informazioni della directory per conto dell'applicazione che ne ha fatto richiesta via TCP/IP. Una richiesta è di solito fatta dal directory client, ed il processo che cerca l'informazione nella directory è chiamato directory server. In generale, i server forniscono un servizio specifico ai client. Le API LDAP forniscono funzioni che permettono alle applicazioni LDAP client di cercare e ottenere informazioni da un Servizio di Directory, così come modificare un'informazione di un entry, se permesso. Il formato e il contenuto dei messaggi scambiati tra client e server deve rispettare il protocollo **LDAP**.

2.2 Active Directory

L'esempio più noto di un servizio di directory è **Microsoft Active Directory (AD)**, che fornisce i metodi per l'archiviazione dei dati della directory e la disponibilità di questi agli utenti e agli amministratori di rete.

2.2.1 Caratteristiche di Active Directory

Active Directory è un **servizio di directory** sviluppato da Microsoft per i domini delle reti Windows, ed è uno strumento fondamentale per l'organizzazione e la gestione degli utenti, dei loro attributi e della loro membership in determinati gruppi di un'organizzazione, gestione di account, di risorse di rete e altro. **Active Directory** funziona come una rubrica per una infrastruttura di rete di un'azienda, infatti essa permette di archiviare informazioni sugli account utente, come nomi, password, numeri di telefono etc.... Essa permette alle aziende di centralizzare servizi di autenticazione e autorizzazione. Essa è creata infatti per controllare se un determinato utente conosca le giuste credenziali (autenticazione) e per determinare a quali file ed a quali applicazioni egli possa accedere in base al suo ruolo e alla sua appartenenza a determinati gruppi (autorizzazione). **Active Directory** ha un ruolo cruciale nel mantenimento dell'ordine e della sicurezza in un'organizzazione ed in una rete. Esso permette agli amministratori autorizzati di gestire utenti, computer, device ed altre risorse da un unico posto centralizzato, rendendo tutta questa gestione più efficiente. L'insieme dei servizi di rete di **Active Directory**, ed in particolare il servizio di autenticazione *Kerberos*, realizzano un'altra delle caratteristiche importanti: il Single Sign-On (SSO). Tramite tale meccanismo un utente, una volta entrato nel dominio ed effettuato quindi il login ad esso da una qualsiasi delle macchine di dominio, può accedere a risorse disponibili in rete (condivisioni, mailbox, intranet etc...) senza dover rieffettuare l'autenticazione. Questo facilita di molto la gestione degli utenti.

e la vita di questi ultimi, a differenza di quanto accade nelle reti peer to peer. **Active Directory** utilizza vari protocolli (principalmente LDAP, DNS, DHCP, Kerberos).

Active Directory fu distribuita come beta nel 1996, proposta per la prima volta con Windows 2000, e in Windows Server 2003 venne rilasciata una versione con nuove funzionalità. Ulteriori miglioramenti sono stati fatti in Windows Server 2003 R2. Active Directory è stata ulteriormente raffinata con Windows Server 2008 e Windows Server 2008 R2 ed è stata ribattezzata in Active Directory Domain Services.

2.2.2 Scopo di Active Directory

Active Directory memorizza informazioni come "oggetti", ovvero ogni risorsa che si trova sulla rete come computer, account utente, contatti, gruppi, unità organizzative etc... Ogni oggetto è caratterizzato da un nome e degli attributi. L'informazione è mantenuta in un data store strutturato che ottimizza le operazioni e fa sì che gli utenti della rete e le applicazioni possano accedere alle informazioni in maniera veloce. Quindi il principale scopo di Active directory è permettere alle organizzazioni di gestire le loro reti in modo sicuro e efficiente.

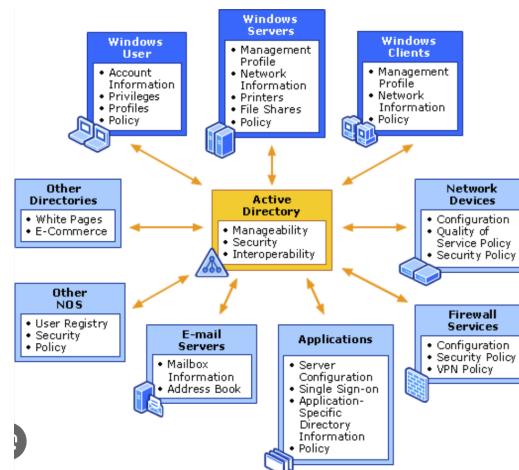


Figura 2.1: Active Directory

2.2.3 Active Directory Domain Services

Gli **Active Directory Domain Services**(AD DS) sono i componenti centrali di Active Directory. Un server sul quale è in esecuzione un **Active Directory Domain Service** è denominato **Domain controller**. Un **domain controller** è un tipo di server che processa le richieste di autenticazione dagli utenti in un dominio. Essi rispondono alle richieste di autenticazione e memorizzano i dati in un **AD DS**. Un domain controller può ospitare anche altri servizi complementari ad un **AD DS**. Active Directory deve avere almeno un Domain Controller. Il concetto importante da capire è che **Active Directory Domain Services** è un framework per il controllo e la gestione di un dominio, e la macchina che gli utenti usano per accedere ad **AD DS** è il **Domain Controller**. **AD DS** è quindi un codice server che permette agli amministratori di gestire e memorizzare informazioni sulle risorse di una rete, in Active Directory in una struttura gerarchica, oltre che permettere la gestione delle autenticazioni e dei controlli di accesso. AD DS fa questo attraverso:

- **Autenticazione Utente:** AD DS autentica gli utenti prima che essi accedano alle risorse di rete, assicurandosi che solo gli individui autorizzati possano entrare in specifiche parti del sistema
- **Archiviazione dei dati:** memorizza i dati della directory, come username, password, numero di telefono, che aiutano a semplificare le operazioni all'interno di un'organizzazione
- **Applicazione delle policy:** Con le GPO(Group Policy Objects) gli amministratori possono applicare policy di sicurezza su più macchine alla volta, garantendo un alto livello di controlli d'accesso

AD DS aiuta a gestire le operazioni di rete garantendo un modo di memorizzare i dati in un organizzazione gerarchica.

2.2.4 Struttura gerarchica di Active Directory

Per funzionare così bene Active Directory include una struttura gerarchica composta da diverse componenti che lavorano insieme per rendere più efficienti le operazioni di rete. Alla sommità della struttura di Active directory c'è la **foresta**. Una **Foresta** ospita tutti gli oggetti, le unità organizzative, i domini e gli attributi. Sotto ad una foresta ci sono uno o più alberi che ospitano domini, unità organizzative, oggetti e attributi. Il cuore della struttura di **Active Directory** sono i **domini**. Il dominio è tipico della varietà dei nomi di Internet, ma non si deve per forza seguire una struttura

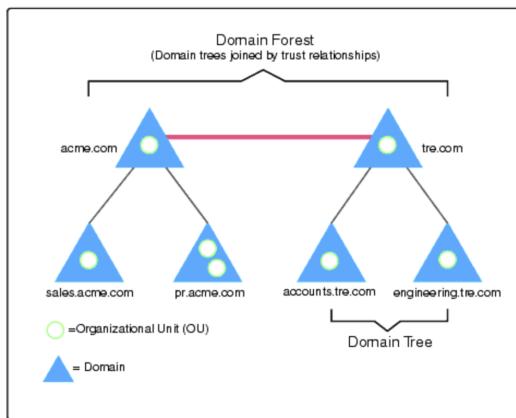


Figura 2.2: Struttura di Active Directory

specifiche come i nomi dei siti web in internet. Possiamo tecnicamente chiamare il nostro dominio come vogliamo. Microsoft raccomanda di usare meno domini possibili e di fare affidamento sulle **Unità organizzative(OU)**. I domini possono contenere più unità organizzative nidificate, consentendo di creare una struttura piuttosto solida e specifica. Un'unità organizzativa è un contenitore di oggetti come utenti, gruppi, devices etc... Una OU può contenere altre OU, permettendo di creare una struttura multilivello , come nella seguente immagine:

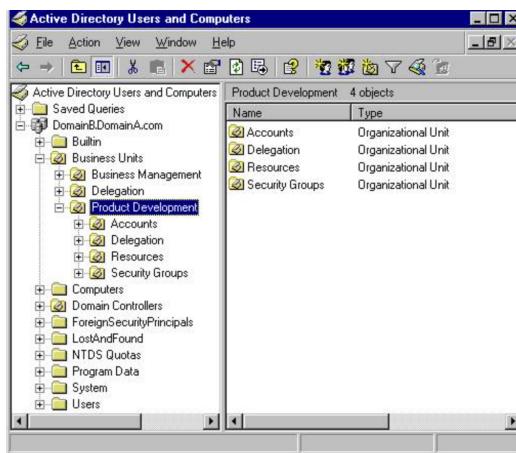


Figura 2.3: Organizational Units di Active Directory

Un altro importante nodo della Struttura di Active Directory sono i **gruppi**. I **gruppi** in Active Directory hanno due principali funzionalità:

- **sicurezza:** un gruppo di sicurezza contiene account che possono essere



Figura 2.4: Groups di Active Directory

usati per accessi di sicurezza. Ad esempio ad un *security group* possono essere assegnati diritti di accedere a particolari directory di un server

- **distribuzione:** i gruppi possono essere usati per mandare informazioni solo a determinati tipi di utenti.

2.2.5 Active Directory Replication

Dal momento che spesso **Active Directory** contiene più **domain controllers** e un utente potrebbe teoricamente attaccarsi ad ogni DC per autenticazione, ogni server ha bisogno di essere aggiornato sui dati. I **Domain Controller** stanno sempre aggiornati replicando i database tra loro stessi. Questo è fatto attraverso un continuo meccanismo di *pull*: un server richiede nuove informazioni da un diverso DC frequentemente. Dopo un cambiamento, il DC inizia una replica dopo aver aspettato circa 5 minuti.

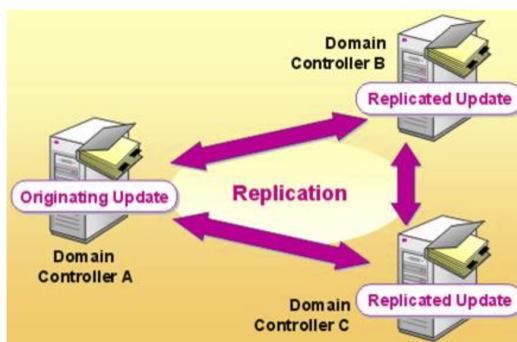


Figura 2.5: Replication in Active Directory

AD usa le *Remote Procedure Calls*(un tipo di richiesta che un client o un altro server manda al server facendogli eseguire un determinato metodo) per la replica e può usare il protocollo SMTP per i cambiamenti a schema o configurazioni.

2.2.6 Active Directory vs LDAP

La principale differenza tra Active Directory e LDAP è che Active Directory è un servizio di directory, mentre LDAP è un protocollo che ha come scopo principale quello di poter comunicare con i servizi di directory LDAP e quindi con Active Directory. LDAP è uno standard cross platform, mentre Active Directory è un software di Microsoft quindi pensato per utenti e applicazioni Microsoft.

Il principale scopo di LDAP è di fare query o modificare directory Server, mentre d'altra parte il principale scopo di Active Directory è di memorizzare informazioni utente, garantire autenticazione e permettere ad amministratori di gestire gruppi, utenti e policy.

In conclusione possiamo dire che LDAP e Active Directotry sono spesso usati insieme, in quanto condividono molti punti in comune e non possono essere viste come soluzioni alternative.

2.3 Protocollo LDAP

LDAP (Lightweight Directory Access Protocol) è un protocollo per la gestione dei servizi di directory che è costruito a livello di applicazione nello stack TCP/IP. Esso garantisce un meccanismo per la connessione, la ricerca e la modifica delle directory.

2.3.1 Storia del protocollo LDAP

Il protocollo LDAP originario è stato ideato nel 1993 da Tim Howes della University of Michigan, Steve Kille di Isode Limited, Colin Robbins di NeXor e Wengyik Yeong di Performance Systems International. Mark Wahl di Critical Angle Inc., Tim Howes, e Steve Kille iniziarono a lavorare nel 1996 a una nuova versione di LDAP, LDAPv3 sotto osservazione dell'Internet Engineering Task Force (IETF). LDAPv3, pubblicato per la prima volta nel 1997, ha sostituito poi LDAPv2. Ulteriori sviluppi alle specifiche di LDAPv3 e numerose estensioni sono state in seguito aggiunte dall'IETF in successive RFC.

L'**IETF** è un organismo internazionale, libero, composto da tecnici specialisti e ricercatori interessati all'evoluzione tecnica e tecnologica di Internet e divisi in gruppi di lavoro che si occupano ciascuno di uno specifico argomento. Il frutto del lavoro di ogni gruppo si traduce in dei documenti denominati **RFC** (Request For Comments) che vengono sottoposti alla **IESG** (Internet Engineering Steering Group) per il loro avanzamento a standard ufficiale. Un **RFC** è un documento pubblicato dalla Internet Engineering Task Force che riporta informazioni o specifiche riguardanti nuove ricerche, innovazioni e metodologie dell'ambito informatico o, più nello specifico, di Internet. Il protocollo **LDAPv3** è specificato nell' **RFC 4511**.

2.3.2 Caratteristiche del protocollo LDAP

Il **protocollo LDAP** è un protocollo a livello di applicazione nato per la gestione e l'utilizzo dei **Directory Service**.

Il protocollo LDAP facilita la gestione e interrogazione di dati e informazioni utili per la gestione degli utenti di una determinata azienda. Per salvare questi dati in maniera più efficiente vengono utilizzati i **Servizi di Directory**. In questi le informazioni e gli attributi riguardanti i vari oggetti come utenti, hardware, applicazioni, stazioni di lavoro o dati di accesso sono raccolti in una struttura gerarchica chiamata **DIT (Directory Information Tree)**. LDAP permette di effettuare ricerche efficienti, apportare modifiche e autenticare gli attributi in un servizio di directory complesso e viene utilizzato come standard per le query e le modifiche secondo il modello client-server. In questo modello per questo si parla spesso di **Server LDAP** quando i server di directory comunicano tramite il protocollo LDAP. Il termine *LightWeight* deriva dal fatto che questa nuova versione del protocollo è la versione leggera del vecchio protocollo **DAP**, Directory Access Protocol, che era troppo complesso per la gestione di aziende che presentavano un alto numero di utenti.

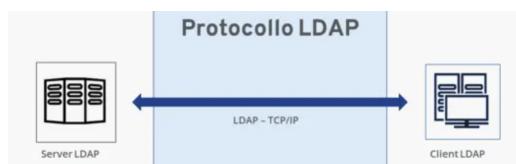


Figura 2.6: Protocollo LDAP

LDAP si basa su uno stack di protocollo TCP/IP e può essere applicato in modo flessibile a qualsiasi sistema di directory. Può servirsi delle porte TCP E UDP per trasmettere i dati. Le porte standard sono la 389 per i trasferimenti dei dati non protetti e la porta 636 per quelli di dati criptati TLS.

2.3.3 Struttura di una directory LDAP

In genere un server LDAP consiste in un **server di Directory** la cui struttura dati corrisponde alle specifiche LDAP e che esegue trasferimenti dei dati tramite l'attuale protocollo LDAPv3.

Nelle configurazioni LDAP si usa una struttura ad albero gerarchica standardizzata (DIT), la quale può essere distribuita su molti server. La gerarchia ad albero è a sua volta divisa o ramificata in diversi livelli politici, geografici e organizzativi rappresentativi come segue:

- Root
- Paesi
- Organizzazioni
- Unità di Organizzazione
- Persone
- Singole Persone (individui, risorse)

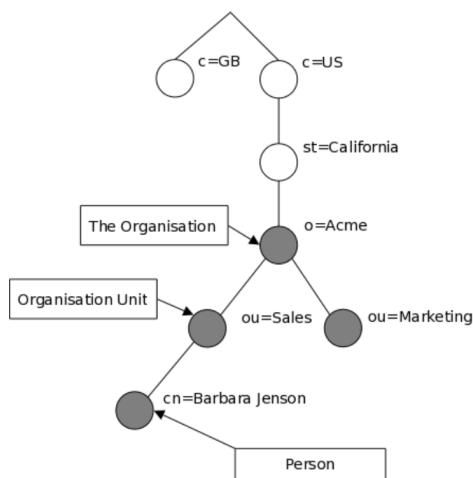


Figura 2.7: Struttura LDAP

L'albero può anche essere adattato sui nomi dei domini Internet.

Le interrogazioni alla directory passano attraverso i Server LDAP, denominati anche Directory System Agent (DSA), che possono distribuire le interrogazioni ad altri Server LDAP, per garantire agli utenti una risposta veloce e efficiente.

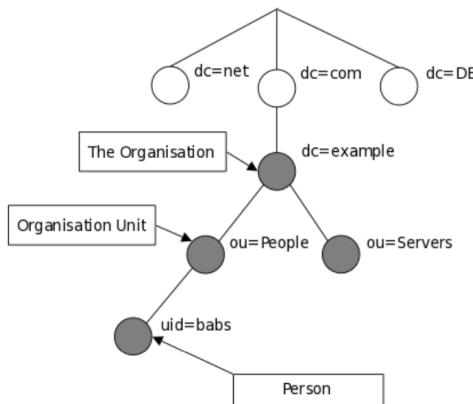


Figura 2.8: Struttura LDAP con domini

LDAP ha un approccio di programmazione orientato agli oggetti che comprende oggetti, classi, ereditarietà e polimorfismo. Ogni voce della directory LDAP (anche detta oggetto LDAP) è costituita da attributi e dalla designazione obbligatoria dell'oggetto **Distinguished Name**. La struttura del nome distinto assomiglia alle convenzioni di denominazione dei file o dei domini. Gli attributi che formano un oggetto possiedono ciascuno un tipo specifico, il quale è identificato da abbreviazioni come **cn, st, sn...**. In più gli attributi possono avere un valore singolo o multiplo a seconda del tipo.

Il protocollo si serve di procedure di accesso specifiche con le quali si comunica al server LDAP chi sta accedendo alla directory tramite la direttiva bind e un nome distino (DN). **BaseDn** può essere utilizzato per definire quali livelli di directory sono ammissibili per la ricerca.

2.3.4 Componenti di LDAP

La struttura leggera di LDAP e l'utilizzo di un DIT consentono la rapida esecuzione di una ricerca LDAP e forniscono risultati immediati. Il **Dit** facilita la navigazione tra i diversi livelli della directory LDAP, per restringere i risultati della ricerca e fornire una risposta ad una query. Una parte fondamentale del DIT è il **Domani Access Component(dc)**, ad esempio `dc=ad,dc=unict`, che utilizza il mapping DNS per individuare i nomi di dominio internet e tradurli in indirizzi IP. La maggior parte degli utenti non è a conoscenza del nome di dominio e/o dell'indirizzo della persona che sta cercando.

2.3.4.1 DNs e RDNs

Un **distinguished name**, anche conosciuto come **DN**, identifica in maniera univoca un entry e la sua posizione nel DIT(directory information tree). Il DN di un entry LDAP è molto simile al path di un file nel filesystem. Un dn è composto da 0 o più elementi chiamati **Relative Distinguished Name** o **RDN**. Ogni RDN è composto da uno o più coppie attributo=valore. Ad esempio "uid=jonh" rappresenta un RDN composto dall'attributo *uid* e dal valore *jonh*. Se un RDN ha più coppie attributi valore, essi sono separati dal segno +, ad esempio *givenName=John+sn=Doe*. Per i DNs con più RDNs, l'ordine degli RDNs specifica la posizione dell'entry nel DIT. Gli RDN sono separati dalla virgola, ogni RDN in un DN rappresenta un livello nella gerarchia in ordine discendente. Ad esempio, il DN "uid=john.doe,ou=People,dc=example,dc=com" ha 4 RDN, ed il DN padre è "ou=People,dc=example,dc=com".

2.3.4.2 Attributi LDAP

Gli attributi ospitano i dati e le informazioni di ogni entry. Ogni attributo ha un tipo, zero o più opzioni, ed un insieme di valori che comprendono il dato attuale. I **tipi degli attributi** sono elementi dello schema LDAP che specificano come gli attributi devono essere trattati dai client e dai server LDAP. Ogni tipo di attributo deve avere un object identifier (**OID**) e zero o più nomi che possono essere usati per riferirsi all'attributo. Essi devono inoltre avere una specifica sintassi, che indica il tipo di dato che può essere memorizzato negli attributi di quel tipo, e un insieme di regole di matching che indicano come i confronti devono essere effettuati tra i valori degli attributi di quel tipo. I tipi degli attributi devono inoltre indicare quando ad un attributo è permesso avere più valori nella stessi entry e se un attributo può contenere dati dell'utente o è solo utile ad alcune operazioni del server. Gli **attributi operazionali** sono attributi di solito usati per la configurazione o per informazioni di stato.

2.3.4.3 Classi di oggetti

Le classi di oggetti sono elementi dello schema LDAP che specificano collezioni di tipi di attributi che possono riferirsi a particolari tipi di oggetti, processi o altre entità. Ogni entry ha un classe d'oggetto che indica che tipo di oggetto un entry rappresenta, ad esempio se un entry contiene informazioni su una persona,un gruppo, un device... e può avere 0 o più classi ausiliarie che permettono ad un entry di avere particolari caratteristiche. Come ogni tipo di attributo, le classi di oggetti devono avere un oid (object identifier)

e possono avere anche uno o più nomi. Le classi di oggetti possono inoltre richiedere un insieme di attributi richiesti, ovvero se un entry appartiene a quella classe deve avere per forza specificati quegli attributi, ed una serie di attributi opzionali.

2.3.4.4 Object Identifier(OIDs)

Un **object identifier** è una stringa che è usata per identificare in maniera univoca diversi elementi nel protocollo LDAP. Un OID consiste in una sequenza di numeri separati da punti (ad esempio “1.2.840.113556.1.4.473”) che serve ad identificare gli elementi dello schema (come ad esempio tipo di un attributi, classe di oggetti, matching rules...).

2.3.4.5 Operazioni di ricerca LDAP

Un’operazione di ricerca LDAP può essere usata per ottenere entries che corrispondono a determinati criteri. Gli elementi di una ricerca LDAP possono includere:

- il *search base dn* che specifica la base del sottoalbero nel quale la ricerca deve essere vincolata. Se esso non è specificato la base sarà la radice della directory
- il *search scope* che specifica la porzione del sottoalbero che deve essere considerata. Ad esempio *baseObject* indica che solo le entry specificate come la base di ricerca devono essere considerate, e non i suoi discendenti ad esempio, oppure *singleLevel* indica che solo i figli immediatamente subito di un entry della base di ricerca devono essere considerati. *WholeSubtree* indica che l’entry specificata come la base di ricerca, e tutti i suoi discendenti, devono essere considerati, mentre *subordinateSubtree* indica che l’entry specificata come la base di ricerca non deve essere considerata, ma tutti i suoi discendenti si.
- La *alias referencing behavior* che indica come il server deve trattare ogni alias che incontra durante il processo di ricerca.
- Il *Size limit* indica il massimo numero di entries che devono essere ritornati da un’operazione di ricerca. Il valore 0 indica che non ci sono limiti.
- Il *Time limit* che specifica il massimo tempo in secondi che il server deve impiegare durante un’operazione di ricerca.

- *Typeonly flag* che se impostato a true indica che le entry che corrispondono al criterio di ricerca devono contenere solo la descrizione degli attributi che contiene, ma non deve contenere quindi i loro valori.
- Il filtro da usare per la ricerca per specificare alcune condizioni che le entry da ritornare devono rispettare.
- Un insieme di attributi che devono essere riportati dall'entry che corrisponde ai criteri di ricerca. Il valore * indica che tutti gli attributi devono essere tornati al client che ha fatto richiesta, il valore + indica che tutti gli attributi operazionali devono essere ritornati, il valore 1.1 indica che nessun attributo deve essere ritornato. Alcuni server supportano pure il simbolo @ seguito dal nome della classe che indica che devono essere ritornati solo gli attributi appartenenti ad una determinata classe.

Quando un server di directory riceve una richiesta di ricerca valida e autorizzata, identificherà tutte le voci nell'ambito specificato che corrispondono al filtro specificato. Tutte queste voci (o almeno quelle che il richiedente ha il permesso di recuperare) verranno restituite al client nei messaggi di voce dei risultati di ricerca.

Un risultato di ricerca include anche un result code, che sarà *success* se la ricerca è stata completata con successo dal server e ritorna le entries che matchano determinate regole, *noSuchObject* se non esiste nessuna entry che corrisponde ai parametri di ricerca, *invalidDNSyntax* che indica che i criteri di ricerca sono errati, *sizeLimitExceeded* che indica che ci sono un numero di entries che corrispondono ai parametri di ricerca maggiore rispetto al size-limit impostato, mentre *timeLimitExceeded* indica che il tempo che il server ha impiegato per un'operazione di ricerca è superiore a quello impostato con *timeLimit*. Il codice di ritorno “*undefinedAttributeType*” indica che il filtro include un tipo che non è specificato nello schema del server, mentre *insufficientAccessRight* indica che chi sta facendo la richiesta non ha i permessi di ricerca e lettura sulla directory .

2.3.4.6 Filtri di ricerca LDAP

I filtri di ricerca sono spesso usati per definire criteri per identificare entries che contengono particolari tipi di informazioni. Ci sono diversi tipi di filtri di ricerca:

- *Presence filter* che possono essere usati per trovare quelle entries dove uno specifico attributo ha almeno un valore. Esempio (cn=*)

- *Equality filters* che possono essere usati per identificare le entries dove uno specifico attributo ha un particolare valore. Esempio (givenName=John)
- *Substring filters* che possono essere usati per identificare quelle entries dove uno specifico attributo ha almeno un valore che contiene una certa sottocorrispondenza. Esempio (cn=Jon*)
- *Greater-or-equal filters* che possono essere usati per identificare entries dove uno specifico attributo ha almeno un valore che è considerato maggiore o uguale ad un determinato valore.
- *Less-or-equal filters* che può essere usato per identificare entries che hanno almeno un valore che è considerato minore o uguale ad un determinato valore.
- *Approximate match filters* che possono essere usati per identificare entries dove uno specifico attributo ha un valore che è approssimativamente uguale ad un determinato valore. Non c'è una definizione precisa di approssimativamente uguale e questa condizione potrebbe cambiare da un server ad un altro

Possono poi essere usate condizioni come AND, OR o NOT per creare filtri di ricerca più complessi e dove si voglia poter specificare più condizioni su più attributi. Ad esempio ”(&(attr1=a)(attr2=b)(attr3=c)(attr4=d))” oppure ”(—(attr1=a)(attr2=b)(attr3=c)(attr4=d))” oppure ”(!givenName=John)”. Come detto nel paragrafo sugli attributi, ogni attributo specifica delle *matching rules* che vengono seguite dalla logica per capire come deve funzionare il confronto tra il valore di un determinato attributo. Ad esempio la regola di matching *caseIgnoreMatch* porterà ad ignorare la differenza minuscolo-maiuscolo del valore di un determinato attributo quando si confrontano due stringhe, mentre la regola *caseExactMatch* porterà ad un confronto case-sensitive tra le stringhe. Ci sono molte regole di matching che un determinato attributo può avere.

2.3.4.7 Modifiche e tipi di modifiche

I client LDAP possono usare richieste di modifiche per fare cambiamenti ai dati memorizzati in un entry. Una richiesta di modifica specifica il dn dell'entry che si vuole modificare e la lista delle modifiche da applicare ad un'entry. Ogni modifica deve avere un tipo di modifica, il nome dell'attributo e un insieme di valori da assegnare. I tipi di modifiche che possono essere fatte ad un entry LDAP sono:

- *add* che indica che uno o più attributi devono essere aggiunti ad una entry. Questo potrebbe essere usato per aggiungere un attributo nuovo o per aggiungere un nuovo valore ad un determinato attributo.
- *delete* che indica che il valore di uno o più attributi dovrebbero essere rimossi da un entry. Se un'operazione di rimozione presenta uno o più valori, solo questi attributi saranno rimossi. Se l'operazione di rimozione non include nessun valore, tutti i valori di quel determinato attributo saranno rimossi.
- *replace* che indica che l'insieme di valori di un determinato attributo specificato devono essere sostituiti da un un nuovo insieme.
- *increment* che indica che il valore di un intero dell'attributo specificato deve essere incrementato di un determinato valore.

Anche un operazione di modifica otterà come risposta uno codice di ritorno, simili a quelli che si ricevono in seguito ad un'operazione di ricerca. In particolare **LDIF**, *LDAP data interchange Format*, è la rappresentazione testuale di un entry LDAP. Esso ha il seguente formato:

```
[id] dn: distinguished_name
attribute_type: attribute_value...
attribute_type: attribute_value...
...

```

Figura 2.9: Formato LDIF

Ogni entry può contenere quanti *attribute type* e quanti *attribute value* vuole, se essi sono definiti nello schema LDAP. Una linea bianca indica la fine di un entry.

2.3.5 Processo di autenticazione LDAP

Il processo inizia quando un utente tenta di accedere dal proprio Pc ad un programma client compatibile col protocollo LDAP, ad esempio un'applicazione di posta elettronica aziendale. Il tentativo di accesso invia una richiesta di autenticazione del DN assegnato all'utente. Il dn viene inviato tramite l'API client o il servizio che avvia il server DSA. Il client stabilisce automaticamente la sessione con il DSA e LDAP utilizza il DN per cercare l'oggetto o il set di oggetti corrispondente nei record presenti nel database LDAP. In questa fase hanno grande importanza gli RDN del dn, perchè forniscano tutti i passaggi della ricerca di LDAP verso il dit per trovare la persona

cercata. Se nel percorso manca un RDN di collegamento sul backend, il risultato ottenuto potrebbe essere non valido. In questo caso, l'oggetto che LDAP sta cercando è l'account utente della persona. Il protocollo potrà autenticare l'utente solo se l'uid e l'userPassword dell'account nella directory corrispondono. Quando l'utente riceve una risposta, che sia valida o meno, il client si disconnette dal server LDAP. Gli utenti autenticati possono quindi accedere all'API e ai relativi servizi, inclusi i file necessari, le informazioni sull'utente e altri dati dell'applicazione, in funzione delle autorizzazioni concesse dall'amministratore di sistema.

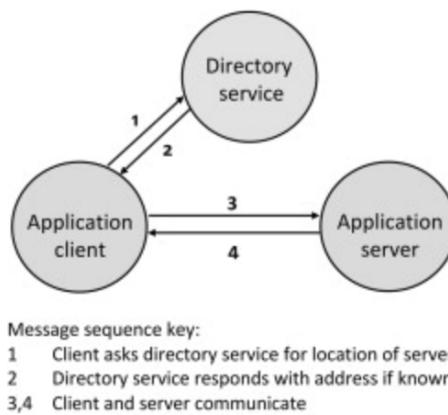


Figura 2.10: Autenticazione LDAP

2.3.6 Differenza tra LDAPv2 e LDAPv3

LDAPv2 e LDAPv3 hanno le stesse finalità ma presentano alcune differenze. LDAv2 diventa uno standard di Internet nel 1995 ed in questa versione viene introdotto il concetto di *Bind Operation*, ovvero l'operazione principale per fare comunicare client e server LDAP. In questa versione l'autenticazione poteva essere fatta semplicemente inviando username e password in chiaro sul canale oppure utilizzando Kerberos per un'autenticazione sicura. La versione LDAPv3 viene specificata direttamente su protocolli di trasporto come TCP ed altri, escludendo del tutto il livello di sessione e presentazione della specifica OSI. Vengono introdotti i meccanismi SASL per rendere i servizi offerti dal server più sicuri. Vi è anche la possibilità di estendere il protocollo introducendo nuove operazioni.

Capitolo 3

Tecnologie Usate

Nel corso di questo capitolo verranno descritti gli strumenti che sono stati scelti per lo sviluppo dell'applicazione.

3.1 OpenLDAP

OpenLDAP è stata una delle scelte più popolari per l'implementazione del protocollo LDAP sin dalla sua nascita (1998).

3.1.0.1 Caratteristiche di OpenLDAP

OpenLDAP è un software che permette agli amministratori di sistema di costruire e manutenzionare una directory LDAP. Esso è un tool da linea di comando e non presenta nessuna user interface, per questo richiede una conoscenza approfondita del protocollo LDAP e della struttura di una directory LDAP. Gli amministratori di sistema possono completare **OpenLDAP** con un'applicazione di terze parti, come *phpLDAPadmin*, che gli permetta di interagire con esso attraverso un interfaccia utente. Poichè **OpenLDAP** è open-source, è facilmente customizzabile e flessibile. Il progetto **OpenLDAP** nacque nel 1998 da Kurt Zelienga. Esso si basava su quanto era stato sviluppato presso l'Università di Michigan, dove si stava portando avanti un progetto di sviluppo ed evoluzione del protocollo LDAP. Nel 2006 poi al progetto si sono uniti Howard Chu e Pierangelo Masarati, che insieme al fondatore rappresentano il team principale di sviluppo.

3.1.0.2 Componenti di OpenLDAP

Il software **OpenLDAP** è formato da 3 componenti principali:

- **slapd**: *stand alone LDAP daemon*, cioè il demone LDAP vero e proprio con i relativi strumenti di controllo. **Slapd** può essere considerato un LDAP directory server che può essere eseguito in diversi sistemi unix.
- le librerie che implementano il protocollo LDAP vero e proprio
- i vari software client per la gestione, l'interrogazione e la manutenzione del database come *ldapsearch*, *ldapadd*, *ldapdelete*...

3.1.0.3 Installazione e configurazione di OpenLDAP

Per poter utilizzare **OpenLDAP** bisogna installare la **OpenLDAP Suite**, che contiene le librerie necessarie al funzionamento del software, un package *openldap-clients* che contiene gli strumenti da linea di comando per la vista e la modifica delle directory ospitate nel server LDAP, un package *openldap-servers* che contiene gli strumenti per configurare ed eseguire un server LDAP (questo package contiene *slapd*).

Prima di poter utilizzare **OpenLDAP**, bisogna settarne la configurazione. Di default la configurazione è salvata nella directory */etc/openldap*. In particolare:

- */etc/openldap/ldap.conf* è il file di configurazione per le applicazioni client che usano le librerie *openldap*
- */etc/openldap/slapd.d* contiene la configurazione di *slapd*

Una volta la configurazione di *slapd* era contenuta nel file */etc/openldap/-slapd.conf*. Adesso essa si trova nella directory */etc/openldap/slapd.d*. Si può convertire la configurazione dal file *slapd.conf* alla configurazione della directory con il comando *slaptest -f /etc/openldap/slapd.conf -F /etc/openldap/slapd.d*.

Il file *slapd.conf* consiste in 3 tipi di informazioni di configurazione: globale, backend e database. Per prima cosa troveremo le informazioni globali, poi le informazioni associate ad un particolare backend, poi le informazioni associate ad un database specifico. Le informazioni che troviamo nella sezione globale possono essere sovrascritte da quelle che troviamo nella sezione backend, le quali a loro volta possono essere sovrascritte da quelle nella sezione database. Le configurazioni più importanti di **OpenLDAP** sono quelle riguardanti questa terza parte. **OpenLDAP** permette l'utilizzo di vari tipi di database come backend, ad esempio *bdb*, *ldap*, *sql*, *mdb*, *ldbm*, *meta*...

La configurazione in slapd consiste in entries LDIF organizzate in una struttura gerarchica e che devono essere modificate attraverso il comando *ldapmodify*, nel caso in cui si voglia cambiare qualche impostazione.

Ad esempio la configurazione del frontend di **OpenLDAP** è memorizzata nel file *etc/openldap/slapd.d/cn=config/olcDatabase={-1}frontend.ldif*, automaticamente definito anche se non specificato.

La directory slapd.d di configurazione contiene le definizioni di schema LDAP che erano prima in */etc/openldap/schema*. E' possibile estendere lo schema usato da **OpenLDAP** inserendo tipi di attributi e classi di oggetti.

3.1.0.4 Lista degli strumenti di OpenLDAP

Per svolgere compiti amministrativi, il package openldap-servers contiene i seguenti strumenti da usare con il servizio slapd:

- *slapcl* permette di controllare l'accesso ad una lista di attributi
- *slapadd* permette di aggiungere entries da un file LDIF ad una directory LDAP
- *slapauth* permette di controllare una lista di id per i permessi di autenticazione e autorizzazione
- *slapcat* permette di prendere le entries da una direcotry LDAP nel formato di default e salvarle in un file LDIF
- *slapdn* permette di controllare una lista di DNs in base alla sintassi disponibile dallo schema
- *slapindex* permette di riindicizzare la directory basata nello schema corrente. Si usa quando si cambiano gli indici della directory nel file di configurazione
- *slappasswd* permette di creare una password utente criptata che può essere usata con ldapmodify o nel file di configurazione slapd
- *slapschema* consente di verificare la conformità di un database con lo schema corrispondente.
- *slaptest* permette di controllare la configurazione del server LDAP

Anche il pacchetto openldap-clients mette a disposizione una serie di strumenti che possono essere usati per aggiungere, modificare, rimuovere entries in una directory LDAP. Essi sono:

- *ldapadd* permette di aggiungere entries ad una directory LDAP da un file o da linea di comando
- *ldapcompare* permette di confrontare un attributo con un'entry della directory LDAP
- *ldapdelete* permette di rimuovere un'entry dalla directory LDAP
- *ldapmodify* permette di modificare un'entry di una directory LDAP
- *ldapmodrdn* permette di modificare il valore RDN di un'entry della directory LDAP
- *ldapsearch* permette di cercare le entries in una directory LDAP
- *ldapurl* permette di decomporre gli url LDAP
- *ldapwhoami* permette di chiedere quale entità sta richiedendo l'operazione corrente

3.1.0.5 Gli OpenLDAP overlays

Gli **overlays** sono componenti software che garantiscono funzionalità analoghe a quelle garantite dai tradizionali backend openldap, ma che possono essere messe sopra alle chiamate backend o dopo una loro risposta per modificarne il comportamento. Gli **overlays** possono essere caricati staticamente o dinamicamente dentro slapd. Essenzialmente gli **overlays** servono a customizzare il comportamento dei backend esistenti senza cambiarne il codice e senza richiederne un altro con complete funzionalità, oltre che a scrivere funzionalità di tipo generico che possono essere utilizzati con diversi tipi di backend. Quasi tutti gli overlay più noti hanno una loro pagina nella documentazione. I più famosi messi a disposizione da OpenLDAP sono:

- **access logging:** questo overlay può registrare l'accesso ad un determinato database.
- **audit logging:** questo overlay può essere usato per registrare tutti i cambiamenti in una dato database in uno specifico file di log.
- **constraint:** questo overlay impone dei vincoli (ad esempio espressioni regolari) su tutti i valori di specifici attributi durante una richiesta di modifica LDAP che contiene il comando ldapadd o ldapmodify. Esso è usato per imporre una sintassi più rigorosa quando essa è troppo generica.

- **dynamic list:** questo overlay permette l'espansione di gruppi dinamici e liste. Invece di avere i membri dei gruppi o la lista di attributi già decisi, questo overlay permette di definire una ricerca LDAP i cui risultati creeranno il gruppo o la lista.
- **memberof :** questo overlay è utile nello scenario in cui è desiderabile per un client essere in grado di determinare di quali gruppi un entry è membro, senza fare un'ulteriore ricerca. Questo overlay aggiorna un attributo (**memberOf**) quando un cambiamento occorre nell'attributo **member** di un entry che è di classe **groupOfNames**. Quindi, esso mantiene la lista dei gruppi alla quale un utente appartiene, modificando solo l'entry del gruppo in questione.
- **proxy cache:** I server LDAP di solito ospitano uno o più sottoalberi di un DIT. I server di replica ospitano copie delle entry ed i cambiamenti sono propagati dal server ai replica server attraverso operazioni di *LDAP sync*. Una LDAP cache è un tipo speciale di replica che mantiene le entry corrispondenti a particolari filtri di ricerca, invece che ad interi sottoalberi.
- **password policies:** questo overlay serve a rinforzare la minima lunghezza di una password, far sì che la password non sia cambiata troppo frequentemente, far sì che una password scada, mantenendo lo storico delle password usato, garantire sicurezza come ad esempio cambiare la password a prima autenticazione etc...
- **return code:** questo overlay è utile per testare il comportamento di un client quando un errore generato dal server avviene, ad esempio codice d'errore etc..
- **rewrite/remap:** questo overlay fa un dn/data rewrite e un mapping degli attributi o delle classi. Esso è usato per garantire una vista virtuale di dati esistenti in remoto, cambiandone il tipo di attributo. Per questo è molte volte usato con il database di tipo *LDAP* che serve per creare un proxy ad un server remoto.
- **sync provider:** questo overlay implementa il supporto per la sincronizzazione del contenuto LDAP e per la persistenza.
- **translucent proxy:** questo overlay può essere usato con una database di backend come mdb per creare un *translucent proxy*. Le entries che provengono da un server LDAP remoto possono avere tutti o alcuni attributi sovrascritti, o nuovi attributi aggiunti, da entry nel database

locale prima che vengono presentati al client. Un'operazione di ricerca è prima popolata con le entry dal ldap server remoto, i cui attributi sono poi sovrascritti con alcuni attributi definiti nel database locale. Gli override locali possono essere popolati con le operazioni di aggiunta o modifica di attributi, il cui uso è ristretto all'utente root del database translucent locale. Un'operazione di confronto farà prima un confronto con gli attributi definiti in locale (se ci sono), prima che un qualsiasi confronto è fatto con i dati nel database remoto.

- **attribute uniqueness:** questo overlay può essere usato con database come mdb per imporre l'unicità di alcuni o tutti gli attributi delle entry in un sottoalbero.
- **value sorting** questo overlay può essere usato con un backend database per ordinare i valori di uno specifico attributo multi valued.

Gli *overlays* possono essere impilati, ovvero più di un overlay può essere istanziato per ogni database. Come conseguenza, ogni funzione di overlay è chiamata quando l'esecuzione dell'overlay è invocata.

3.2 Tecnologie per lo sviluppo

La web app implementata utilizza alcune delle tecnologie alla base dello sviluppo web degli ultimi anni, come HTML e CSS che sono alla base della creazione dell'interfaccia grafica dell'applicazione, e Javascript, utile per lo sviluppo sia frontend che backend (grazie all'utilizzo di Node.js).

3.2.0.1 HTML

L'**HTML** (Hypertext Markup Language) è un insieme di simboli di markup e di codice inserito in un file che deve essere mostrato in un browser. Il linguaggio di markup dice al browser come deve disporre le parole, le sezioni etc... di una pagina web. Il linguaggio HTML facilita la creazione di siti web. Esso ha codice e sintassi così come ogni altro linguaggio, è relativamente facile da comprendere e sta diventando sempre più potente.

Il linguaggio HTML consiste in una serie di codici corti scritti all'interno di un file di testo. Questi sono i **tag** HTML, il cuore del linguaggio. Il testo è salvato in un file HTML e viene letto dal browser, che lo traduce in forme visibili in base a come il programmatore ha scritto la pagina HTML. I **tag** sono quelli che separano testo normale dal codice HTML. Essi sono parole che si trovano tra `<>` e che permettono la definizione di elementi grafici,

tabelle, link, immagini etc... che vogliamo appaiano nella pagina web. **Tag** diversi permettono di creare elementi diversi.

Un esempio banale di tag è il tag `<div>`, che insieme alla sua parte finale `</div>` permette la creazione di una sezione di testo con all'interno il testo o altri elementi HTML definiti nel codice tra il tag iniziale ed il tag finale.

3.2.0.2 CSS

Il **CSS** (Cascading Style Sheets) è un linguaggio di stile che permette di specificare come le pagine HTML devono essere presentate dal browser all'utente. Il file che il browser solitamente riceve da un server è un file che è strutturato attraverso il linguaggio di markup **HTML**. Tuttavia presentare un documento HTML all'utente non permette di creare grafiche adatte e facili da comprendere all'occhio umano. Il **Css** può essere usato per creare fogli di stile che permettano ad esempio di cambiare il colore o la taglia del testo, il colore di un paragrafo etc... Esso quindi può essere usato per creare il layout di una pagina web, oltre che per creare effetti come animazioni etc.. Il **Css** è una lingua basata sulle regole, con le quali definiamo gruppi di stile che devono essere applicati a determinati elementi o gruppi di elementi HTML nella nostra pagina web. In particolare attraverso i **selettori** definiamo a quali elementi deve essere applicata una determinata regola di stile che ne modificherà determinate rappresentazioni (ad esempio attraverso la regola `color` possiamo modificare il colore del testo di quegli elementi etc...)

Una semplice regola **Css** è creata da

```
selettori{  
regola : proprietà  
}
```

3.2.0.3 Javascript

Javascript (JS) è un linguaggio di programmazione usato per creare contenuto interattivo e dinamico nel web, oltre che per creare applicazioni eseguibili in modo sicuro nel browser.

Javascript è un linguaggio leggero con una sintassi molto vicina a Java, interpretato ed orientato agli oggetti, conosciuto per lo più come linguaggio di script per la creazione di pagine web, ma utilizzato in molti ambienti non-browser come `node.js`. Con linguaggio di script intendiamo un linguaggio di programmazione usato per automatizzare processi che gli utenti dovranno fare da se step by step. **Javascript** viene eseguito dai browser grazie ad un particolare engine built-in. Questo vuol dire che i comandi js possono

essere messi all'interno di un documento HTML, ed il browser sarà in grado di interpretarli.

Javascript viene di solito inglobato nella pagina web stessa oppure incluso come file .js.

Javascript è sia un linguaggio **client-side** che un linguaggio **server-side**. Lato client è usato per aggiungere interattività ad un sito web, in modo da trasformare una pagina web da statica a dinamica.

Lato server è invece usato per creare il backend di un'applicazione (ovvero creare le API che permetteranno di garantire determinati servizi a chi lo richiede).

3.3 Tecnologie di supporto

3.3.1 Node.js e Npm

3.3.1.1 Node.js

Node.js è un ambiente di esecuzione Javascript open-source e cross-platform. Esso permette di avere un ambiente che permetta di eseguire un programma scritto in Javascript. Browser come Chrome e Firefox hanno engine built-in al loro interno che permettono l'esecuzione di Js e la costruzione del front-end dinamico di applicazioni web. **Node.js** permette di avere un ambiente di esecuzione fuori dal browser. **Node.js** è sempre costruito sull'engine *Chrome V8 Javascript*. Il suo scopo principale è la possibilità di creare il backend delle applicazioni usando lo stesso Javascript con il quale è familiare scrivere il frontend delle applicazioni web.

3.3.1.2 Npm

Uno dei maggiori fattori di Node.js è **Npm**, il suo *package manager*. Esso è anche conosciuto come "Ninja Pumpkin Mutants". Un *package manager* è una collezione di strumenti software che automatizzano il processo di installazione, aggiornamento, configurazione e rimozione di software (incapsulato in *packages*) da un determinato progetto.

Npm consiste in due parti principali:

- una CLI (command line interface) che permette di pubblicare o scaricare pacchetti
- una repository online che ospita i pacchetti Javascript

Ogni progetto in Javascript può essere visto come un *npm package* con il suo personale pacchetto di codice ed un file ***package.json*** che descrive il progetto. Possiamo pensare al file ***package.json*** come un'etichetta del nostro progetto che contiene tutti i pacchetti utilizzati e le loro dipendenze. Il file ***package.json*** sarà generato quando viene eseguito il comando ***npm init***, che permette di inizializzare un progetto Javascript/Node.js.

Il file ***package-lock.json*** è invece generato dal comando ***npm install*** ed è una "tabella degli ingredienti" che descrive le esatte versioni delle dipendenze e che non è destinato alla lettura degli sviluppatori.

3.3.2 Moduli Javascript utilizzati

All'interno dell'applicazione sono stati utilizzati diversi pacchetti Javascript, installati grazie al gestore di pacchetti **Npm**.

Express.js è il framework backend per Node.js più popolare e fa parte dell'ecosistema Javascript. Esso offre strumenti solidi per lo sviluppo di applicazioni backend scalabili. Fornisce una serie di strumenti per applicazioni web, la gestione delle richieste e delle risposte HTTP, il routing e il middleware per la creazione e la distribuzione di applicazioni su larga scala e di livello aziendale. La sua principale funzionalità è quella di poter sviluppare gli endpoint (API) e i sistemi di routing di richieste alle applicazioni web.

Express fornisce meccanismi per:

- scrivere gestori per le richieste con diversi verbi HTTP a diversi url path dell'applicazione web.
- integrare con engine di rendering di view per generare risposte con dati da inserire dentro un template.
- settare impostazioni comuni per le applicazioni web come porte per la connessione o la locazione dei template usati per il rendering delle risposte.
- aggiungere "middleware" che vengono eseguiti nel mezzo della pipeline di gestione delle richieste.

Insieme ad **Express**, gli sviluppatori hanno creato pacchetti per gestire molti dei problemi che si presentavano nella creazione delle applicazioni web, come gestione di cookie, sessioni, login etc...

Uno dei middleware che viene utilizzato all'interno dell'applicazione è il pacchetto *express-session*. Questo middleware usato in Express permette di creare sessioni nell'applicazione web. Esso memorizza i dati di sessioni server-side, usando una serie di opzioni di memorizzazione, per tenere traccia delle

attività di un utente che fa richieste all’applicazione.

Un ulteriore pacchetto che viene utilizzato all’interno dell’applicazione è il package *ldapjs-client*, che fornisce oggetti e metodi per poter interfacciarsi col server **OpenLDAP** e poter fare richieste di ricerca, modifica o aggiunta di gruppi o utenti.

3.3.3 EJS (Embedded Javascript)

EJS è un popolare engine di template per Node.js che permette di generare HTML markup con semplice Javascript. Esso è particolarmente utile per la creazione di pagine web dinamiche, in quanto permette di embeddare la logica di Javascript direttamente all’interno dell’HTML. Esso permette di inserire dati all’interno dell’HTML lato client in modo da produrre l’HTML finale in base ai dati che provengono dal server. L’utilizzo si basa su del codice EJS che viene inserito all’interno di file .ejs, sono semplici file HTML dove viene inserito del codice **Embedded Javascript** tra i tag <> . Ejs permette di definire il contenuto di determinate variabili che devono essere dinamicamente valorizzate all’interno di un file ejs in base al tipo di richiesta o ai parametri che arrivano. Affinchè si possa usare l’engine EJS bisogna installarlo all’interno di un progetto Node tramite il gestore dei pacchetti Npm.

3.3.4 Container e Docker

Negli ultimi anni **Docker** è diventato uno strumento sempre più popolare per la gestione e la distribuzione di applicazioni in **container**.

3.3.4.1 Caratteristiche dei container

Un **container** è un’unità software che racchiude l’applicazione e tutte le sue dipendenze, isolandola dal sistema operativo e dall’hardware sottostanti. In particolare esso permette di impacchettare il codice dell’applicazione insieme alle librerie e alle dipendenze necessarie in modo che possa essere eseguito ovunque.

I container utilizzano al meglio una forma di virtualizzazione del sistema operativo in cui le funzionalità del kernel di esso possono essere utilizzate per isolare i processi e controllare le quantità di cpu, memoria e disco a cui tali processi possono accedere. I **container** sono piccoli, veloci e portatili perché a differenza di una *macchina virtuale* non hanno bisogno di includere un sistema operativo guest in ogni istanza e possono invece semplicemente utilizzare le funzionalità e le risorse del sistema operativo host. Ci sono

grandi differenze tra **container** e **macchine virtuali**. Nella virtualizzazione viene utilizzato un *hypervisor* per virtualizzare l'hardware fisico. Ogni macchina virtuale contiene quindi un sistema operativo guest ed una copia dell'hardware necessario per l'esecuzione del sistema operativo. I **container** invece di virtualizzare l'hardware sottostante virtualizzano il sistema operativo con i suoi dispositivi in modo che ogni singolo container contenga solo l'applicazione e le relative librerie e dipendenze. L'assenza del sistema operativo guest è il motivo per cui i container sono così leggeri e portatili. I **container** permettono di creare ad ogni processo l'illusione di essere l'unico processo in esecuzione sul sistema. I container sono facili da automatizzare e gestire soprattutto grazie all'utilizzo di strumenti di orchestrazione come **Docker** o **Kubernetes**.

3.3.4.2 Docker

Docker è una piattaforma per la creazione, la distribuzione e la gestione di applicazioni in **container**. Un **container** è un'unità software che racchiude l'applicazione e tutte le sue dipendenze, isolandola dal sistema operativo e dall'hardware sottostante. Questo permette alle applicazioni di essere eseguite in modo affidabile e consistente su qualsiasi ambiente, indipendentemente dalla differenza tra i sistemi operativi, le librerie e le versioni di runtime.

Docker utilizza un'architettura client-server, dove il client comunica con il daemon **Docker** tramite un API RESTful. Il daemon **Docker** gestisce il ciclo di vita dei container, inclusa la creazione, l'avvio, la sospensione, la ripresa e la rimozione. Per creare un container è necessario definire un' **immagine docker**. Un'**immagine docker** è una sorta di template che descrive come un container deve essere costruito. L'immagine può contenere tutti i file dell'applicazione, le dipendenze e le configurazioni necessarie per far funzionare l'applicazione in un ambiente containerizzato. Una volta creata l'immagine, è possibile utilizzarla per creare una o più **container**. Un'**immagine docker** viene creata grazie ad un file denominato **Dockerfile**.

Un **Dockerfile** è un file manifest che descrive l'immagine di base sul quale costruire i vari layer e gli elementi da installare su di essa, le variabili d'ambiente presenti nel container al momento della creazione e eventualmente il comando da lanciare all'avvio del container. Un **Dockerfile** ospita una sequenza di istruzioni (come FROM, RUN, ADD, CMD, ENV...) che via via costruiscono l'immagine. In particolare tali comandi sono:

- **FROM:** obbligatoria, va a definire l'immagine di base su cui si va poi a costruire la propria.

- **RUN:** esegue un comando di shell aggiungendo un nuovo layer all’immagine di base, ed è utile per l’installazione di software e package aggiuntivi necessari per l’esecuzione dell’applicazione desiderata.
- **COPY:** permette di copiare dei file locali sull’immagine nella posizione specificata.
- **ENV:** da la possibilità di definire delle variabili d’ambiente utilizzabili all’interno del container.
- **VOLUME:** crea un punto di mount per il path specificato, dichiarandolo esterno al container, appartentente all’host oppure ad un altro container.
- **CMD:** permette di eseguire un comando di shell a runtime, cioè una volta che il container viene lanciato. Lo scopo di questa istruzione è quella di fornire un’istruzione di default che il container dovrà eseguire.
- **ENTRYPOINT:** lo stesso scopo di CMD, definisce il programma da eseguire all’avvio del container. La differenza principale consiste nel fatto che un comando passato da riga di comando con *docker run* può sovrascrivere CMD ma non l’ENTRYPOINT.
- **EXPOSE:** indica quali porti possono essere raggiungibili dall’esterno.

Una volta creato il ***Dockerfile*** è possibile costruire l’immagine di Docker utilizzando il comando *docker build*. Questo processo scaricherà le dipendenze e creerà l’immagine. Dopo aver creato l’immagine è possibile eseguire il container attraverso il comando *docker run*. Questo creerà un’istanza del container e lo avvierà.

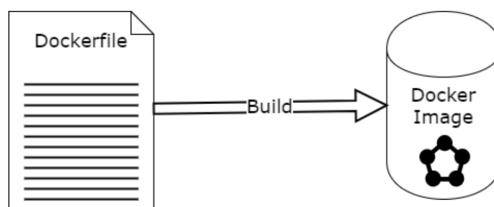


Figura 3.1: Dockerfile

Oltre a questi comandi base, Docker offre un’ampia gamma di comandi per la gestione dei container come *docker ps* per visualizzare i container in esecuzione, *docker stop* per fermare un container o *docker rm* per eliminarlo.

In sintesi quindi l'utilizzo di **Docker** può semplificare la gestione delle applicazioni, consentendo di eseguirle in un ambiente isolato e facilmente trasportabile.

3.3.4.3 Docker container networking

Molte volte i container hanno bisogno di poter collaborare tra loro all'interno di una stessa rete isolata per poter usufruire una dell'altra dei servizi offerti. Un esempio basilare potrebbe essere il fatto che un'applicazione in esecuzione all'interno di un determinato container abbia bisogno magari di un servizio di database all'interno di un altro container. **Docker Compose** è uno strumento per definire ed eseguire più container **Docker**. Esso permette di creare un'applicazione con più container Docker, reti e volumi con una semplice configurazione che può iniziare e terminare un'applicazione con un singolo comando.

Per poter far questo definiamo la configurazione della rete e dei container attraverso un file YAML. Un semplice esempio di un file compose è:

```

1. services:
2.   web:
3.     build: .
4.     ports:
5.       - "8000:8000"
6.   db:
7.     image: postgres
8.     ports:
9.       - "8001:5432"

```

Figura 3.2: File compose

In questo esempio stiamo definendo due servizi:

- un container che parte da un Dockerfile nella directory utente da dove eseguiamo il file compose
- un database che parte da un immagine pre esistente

Il concetto base del **Docker Compose** è la possibilità di poter creare una rete di container. **Docker** offre tre reti di default:

- **bridge:** rappresenta l'interfaccia docker0 presente sugli host che ospitano **Docker**. Di default i containers appartengono a questa rete, ed inoltre i containers saranno in grado di comunicare tra loro utilizzando l'indirizzo IP a loro assegnato.

- **none:** i containers appartententi a questa rete non avranno nessuna interfaccia di rete se non quella di loopback.
- **host:** i container aggiunti alla rete host vedranno l'interno stack di rete visto dal docker host.

Docker da inoltre la possibilità di creare nuove reti o sottoreti a proprio piacimento. I container quindi appartengono alla rete e possono localizzare altri container a partire dal loro nome. All'interno del compose possiamo definire anche su quale porta ascolta quel determinato container, mappandola con la porta della macchina locale su cui stanno venendo eseguiti (se un pacchetto arriva sulla porta 8000 della macchina, verrà inviato alla porta 8000 del container). Possiamo creare una rete vera e propria alla quale appartengono tutti i container predefiniti attraverso i comandi nel file yaml come

```

25   networks:
26     dmi:
27       name: localLan
28       driver: bridge
29       ipam:
30         config:
31           - subnet: 10.0.200.0/24
32         # external: true

```

Figura 3.3: Docker network

Con la possibilità di indicare che un container appartiene ad una determinata rete ed ha un certo indirizzo ip all'interno di quella rete.

```

13   app:
14     image: app
15     hostname: app
16     container_name: app
17     ports:
18       - 8083:8082
19     command: npm start
20     networks:
21       dmi:
22         ipv4_address: 10.0.200.21

```

Figura 3.4: Docker network

Capitolo 4

Realizzazione del progetto

In questo capitolo verrà descritta l'implementazione del progetto, oggetto di questa tesi. Il progetto creato consiste nella configurazione di un OpenLDAP Server, che funziona da proxy all'Active Directory d'ateneo, gestibile tramite un'applicazione web (*Traslucent Proxy Admin*) che permette la creazione di nuovi gruppi, alla quale aggiungere gli utenti già registrati nella directory d'ateneo, e la gestione di determinati attributi locali. In questa sezione si parlerà quindi di come è configurato l'OpenLDAP Server e di come è stata implementata l'applicazione web.

4.1 Configurazione del container Proxy OpenLDAP

Il primo passo fatto nello sviluppo dell'app è la creazione del Dockerfile per la costruzione dell'immagine del container che conterrà al suo interno un'istanza del server LDAP, configurato per eseguire le funzionalità previste. Il Dockerfile per la costruzione dell'immagine del container Proxy è il seguente:

```
↳ Dockerfile > ...
1  FROM debian:10-slim
2
3
4  RUN apt update
5  RUN DEBIAN_FRONTEND=noninteractive apt-get install -y slapd ldap-utils
6  RUN apt-get install nano
7  RUN apt-get install coreutils
8  RUN apt-get install -y gettext
9
10 RUN cd /etc/ldap/slapd.d && rm -Rf *
11 COPY ./slapd_template.conf /etc/ldap/slapd_template.conf
12
13 RUN chmod 700 /var/lib/ldap
14 RUN cd /etc/ldap && chmod 777 -Rf slapd.d
15
16 ENV PATH $PATH:/etc/ldap
17 ENV ROOT_DN_CONFIG "cn=admin,cn=config"
18 ENV ROOT_PW "password amministratore"
19 ENV LOCAL_ATTRIBUTES "member,memberOf,objectClass"
20 ENV REMOTE_ATTRIBUTES "member,memberOf"
21 ENV BIND_DN "dn amministratore"
22 ENV BIND_PW "password amministratore"
23 ENV URI "uri server ldap"
24
25
26 ADD manager.sh /etc/ldap/manager.sh
27 WORKDIR /etc/ldap
28 ADD addMemberOf.ldif .
29 ADD composeStructure.ldif .
30 ADD composeStructureTraslucent.ldif .
31 ADD compose2.ldif .
```

Figura 4.1: Dockerfile OpenLDAP Proxy

Le righe di codice di questo Dockerfile:

- 1: L’immagine di partenza è l’immagine *debian:10-slim*. Essa, presente nel Docker Hub, contiene esclusivamente una versione virtualizzata del sistema operativo Debian, una delle più famose distribuzioni Linux, con la mancanza di alcuni file molto spesso inutili per il corretto funzionamento di un container.
- 4-8: In queste righe si vanno a costruire mano mano i layers dell’applicazione. Alla riga 4 viene eseguito il comando *apt update*. *Apt* è il gestore pacchetti di Debian, ed il comando *update* serve ad aggiornare la lista dei pacchetti disponibili e le loro versioni nella lista del gestore. Nella riga 5 vengono installati il demone *slapd* e *ldap-utils*, che permettono di avere le librerie ed il software OpenLDAP che servirà da LDAP server. L’opzione *DEBIAN-FRONTEND=noninteractive* permette di specificare che non si vuole avere nessun tipo d’interazione durante l’installazione dei pacchetti, mentre con *-y* indichiamo che a qualsiasi tipo di domanda fatta dal gestore di pacchetti durante l’installazione venga risposto in maniera affermativa. Alla riga 6 viene installato il tool *nano*, un editor da terminale per i sistemi operativi Unix, alla riga 7 viene installato *core-utils*, un pacchetto GNU che contiene implementazione

di molti strumenti base come cat, ls, rm... Alla riga 8 viene installato il tool *gettext* che è utile per tradurre una stringa di testo nel linguaggio nativo utente.

10-11: In queste righe ci si sposta nella cartella */etc/ldap/slapd.d* e si rimuovono tutti i file di configurazione di default già presenti. Tale cartella è la cartella che contiene tutti i file di configurazione del server OpenLDAP. Alla riga 11 viene fatta una copia del file di configurazione di template all'interno della cartella */etc/ldap*. Questo template è quel file a partire dal quale verrà creata la configurazione del server OpenLDAP (ovvero verranno creati tutti i file di configurazione nella cartella appena svuotata).

13-14: In queste righe grazie al comando *chmod* vengono cambiati i permessi alla cartella */var/lib/ldap*, dove saranno memorizzati i database del server LDAP, settandoli al livello lettura-scrittura-esecuzione per il proprietario e a nulla per gli altri. Alla riga 14 ci si sposta nella cartella */etc/ldap* e vengono cambiati i permessi ricorsivamente a tutte le sottocartelle e ai file di *slapd.d*, la cartella dove verrano inseriti i file di configurazione.

16-23: In queste righe vengono settate le variabili d'ambiente che saranno presenti all'interno del container al momento dell'esecuzione. In particolare queste variabili d'ambiente sono quelle che verranno utilizzate all'interno del file *slapd.conf* per rendere la configurazione del server OpenLDAP customizzabile dal Dockerfile.

26-31: In queste righe viene copiato il file shell *manager.sh* che contiene i comandi col quale deve partire il container all'interno della cartella */etc/ldap*. Con l'istruzione WORKDIR */etc/ldap* si indica che quella sarà la cartella iniziale di lavoro del container, nella quale con le istruzioni alle righe 28, 29 e 30 vengono copiati i tre file che serviranno per aggiungere qualcosa di default alla directory LDAP

Il file shell *manager.sh* è così costruito:

```
$ manager.sh
1  #!/bin/sh
2  envsubst < "/etc/ldap/slapd_template.conf" > "/etc/ldap/slapd.conf"
3
4  mkdir -p /cache
5  /usr/sbin/slapttest -f /etc/ldap/slapd.conf -F /etc/ldap/slapd.d
6  slapd -F slapd.d -d 0 &
7
8  sleep 6
9
10 ldapadd -x -D "..." -w $ROOT_PW -f composeStructure.ldif
11 ldapadd -x -D "..." -w $ROOT_PW -f composeStructureTraslucent.ldif
12 ldapadd -x -D "..." -w $ROOT_PW -f compose2.ldif
13
14 echo "Proxy started"
15
16 tail -f /dev/null
17
```

Figura 4.2: File manager.sh

Questo file contiene una sequenza di comandi Unix che verranno eseguiti dal container nella fase iniziale. In particolare il primo comando *envsubst* cerca il pattern di determinate variabili d'ambiente definite nel file di template e sostituisce queste variabili con il loro valore. Quindi nel file di configurazione di template dopo questo comando troveremo le variabili d'ambiente sostituite con il valore settato nel Dockerfile. Alla riga 4 viene creata una cartella cache utile al server OpenLDAP. Il comando alla riga 5 è uno dei comandi più importanti della fase di configurazione. Esso permette di convertire la configurazione statica definita nel fine *slapd.conf* ed ormai deprecata nella configurazione moderna, ovvero in un insieme di file di configurazione memorizzati nella cartella *slapd.d*. Alla riga 6 viene avviato il daemon Slapd indicando di seguire le configurazioni definite nella cartella. Con l'opzione *-d 0* indichiamo di non voler nessuna indicazione di debug. La lettera *&* in un file shell indica che il comando pre scritto deve essere eseguito in background e devono essere quindi eseguiti i comandi successivi. Alla riga 8 viene inserita una pausa di 6 secondi, per attendere che il daemon sia correttamente partito, poi tramite i comandi OpenLDAP creiamo la struttura iniziale del nostro DIT, inserendo in ordine le seguenti entry:

```

dn: dc=unict,dc=ad
objectClass: top
objectClass: domain
dc: unict

dn: ou=Gruppi Locali,dc=unict,dc=ad
ou: Gruppi Locali
objectClass: organizationalUnit

dn: ou=Studenti Locali,dc=unict,dc=ad
ou: Studenti Locali
objectClass: organizationalUnit

```

Figura 4.3: DIT 1

```

1
2 dn: ou=Studenti,dc=unict,dc=ad
3 ou: Studenti
4 objectClass: organizationalUnit

```

Figura 4.4: DIT 2

```

dn: ou=Gruppi Studenti,dc=unict,dc=ad
ou: Gruppi Studenti
objectClass: organizationalUnit

```

Figura 4.5: DIT 3

E' stata quindi creata la struttura iniziale del DIT ospitato dalla nostra directory LDAP.

Alla riga 12 si ha un semplice comando di debug che permette di avvertire che il proxy è correttamente partito . Il comando finale serve per mandare un EOF al programma in modo che esso non attenda input da linea di comando. A questo punto il server LDAP è partito e si può interagire con esso.

4.2 Configurazione del Container App

Il secondo passo fatto nello sviluppo dell'app è la creazione del Dockerfile a partire dal quale verra creata l'immagine di un container che possa fornire un ambiente contenente tutti i pacchetti e le librerie necessarie per poter far eseguire un'applicazione Javascript su Node.js. Il Dockerfile per la costruzione di questa immagine è:

```
app_node > 📄 Dockerfile > ...
1   FROM node:22-slim
2
3
4   WORKDIR /tmp/app
5   ADD appjs .
6
7   WORKDIR /tmp/app/appjs
8   RUN npm install express
9   RUN npm install express-session
10  RUN npm install ejs
11  RUN npm install ldapjs-client
12  RUN npm install
13
```

Figura 4.6: Dockerfile WebApp

- 1: Alla riga 1 viene specificata qual è l'immagine di partenza. In particolare l'immagine di partenza *node:22-slim* parte a sua volta da un'immagine Debian che contiene al suo interno già Node ed il gestore di pacchetti Npm. La versione slim è stata progettata per mettere a disposizione un'immagine che mantenga solo gli strumenti necessari per lo sviluppo di un'applicazione in Node e nient'altro.
- 4-5:** Alla riga 4 si indica che la cartella di lavoro deve essere */tmp/app*, ovvero la cartella dove verranno eseguiti tutti i comandi successivi. In particolare alla riga 5 viene copiata la cartella locale dove è contenuta l'app all'interno della cartella di lavoro
- 7-12:** Alla riga 7 ci si sposta in un'altra cartella, la cartella dove è contenuta l'app copiata precedentemente (*/tmp/app/appjs*) e qui, grazie al gestore dei pacchetti *npm*, vengono installati tutti i *packages* chesaranno utili nell'implementazione dell'applicazione. In particolare vengono installati *express* ed *express-session* utili per l'implementazione di un ambiente Server, *ejs* utile per avere l'engine di template delle pagine HTML ed *ldapjs-client* che è la libreria che ci permette di interagire con il container OpenLDAP. L'ultimo comanda installa tutti i pacchetti definiti dalle dipendenze nel file *package.json*.

Al lancio del container si avrà così un'applicazione funzionante, in quanto l'applicazione è già sviluppata nella cartella *appjs* e montata all'interno del container, che contiene tutti gli strumenti necessari richiesti per poter funzionare.

4.3 Configurazione della rete di container

Il container contenente l'applicazione web ha bisogno di poter rintracciare e comunicare con il container contenente il server OpenLDAP in modo da potergli fare query, poter fare modifiche o aggiornamenti. Per questo viene creata una rete docker di container, attraverso l'utilizzo del *Docker Compose*. Per poter creare una rete di container viene definito un file compose.yml, che contiene le istruzioni utili al lancio di ogni singolo container e alla costruzione della sottorete Docker. Il file compose.yml del progetto è:

```

compose.yml
1  version: "1.0"
2  services:
3    proxy:
4      build: .
5      hostname: proxy
6      container_name: proxy
7      ports:
8        - 389:389
9      command: manager.sh
10     networks:
11       dmii:
12         ipv4_address: 10.0.200.20
13
14   app:
15     build: ./app_node
16     hostname: app
17     container_name: app
18     ports:
19       - 8083:8082
20     command: npm start
21     networks:
22       dmii:
23         ipv4_address: 10.0.200.21
24     depends_on:
25       - proxy
26     networks:
27       dmii:
28         name: dmii
29         driver: bridge
30         ipam:
31           config:
32             - subnet: 10.0.200.0/24

```

Figura 4.7: File Compose per la rete di container

- 1: La versione indica la versione del formato del file compose che si sta creando. Essa è tipicamente la prima riga del file e specifica quali funzionalità e quale sintassi è disponibile.
- 2: L'istruzione *services* viene utilizzata all'interno del file compose come elemento top-level, come una mappa le cui chiavi sono rappresentazioni dei nomi di servizi e i cui valori sono le definizioni delle caratteristiche dei servizi. La definizione di un servizio contiene le configurazioni che sono applicate ad ogni container ("servizio") che sta venendo lanciato.

Ad esempio in questo caso dopo services si va a definire il servizio con chiave *proxy* ed il servizio con chiave *app*.

3-12: In queste righe si definiscono le configurazioni del container Proxy, che conterrà il server OpenLDAP. Alla riga 4 viene indicato il Dockerfile a partire dal quale deve essere costruita l'immagine del container (il demone Docker costruirà un'immagine a partire dal Dockerfile presente in quella cartella). Alla riga 5 viene dato un nome al container. In particolare l'*hostname* sarà il nome che il container conoscerà di se stesso, ovvero , nel caso succedesse un evento non usuale durante l'esecuzione, questo nome sarà mostrato nel prompt dei comandi. Alla riga 6 con *container-name* viene dato un nome al container, in modo che Docker non gliene assegna uno casuale. Attraverso questo nome il container sarà riconoscibile anche agli altri container della rete, quindi può essere usato come nome DNS all'interno della rete dei container. Alle righe 7-8 si specifica il mapping delle porte. In particolare si indica che qualsiasi pacchetto che arriva alla porta 389 della nostra macchina locale deve essere inoltrato alla pagina 389 del container proxy. L'istruzione command alla riga 9 specifica il comando che verrà eseguito all'interno del container quando esso verrà lanciato. In questo caso viene lanciato il file shell che setterà le configurazione e creerà la struttura del DIT. Con le istruzioni alle righe 11 e 12 viene specificato a quale rete di container il proxy dovrà appartenere. Il proxy dovrà appartenere alla rete *dmii* e dovrà avere come indirizzo ip l'indirizzo 10.0.200.20.

13-24: In queste righe si vanno a definire le configurazioni del container che conterrà l'applicazione. Verrà costruita un'immagine a partire dal Dockerfile presente nella cartella *app_node*. Viene dato al container il nome di "app", col quale sarà rintracciabile anche dagli altri container della rete. Alla riga 17 è presente il mapping delle porte, indicando che qualsiasi pacchetto che arriverà alla macchina locale alla porta 8083 dovrà essere inoltrato alla porta 8082 del container. Alla riga 19 viene indicato il comando che verrà eseguito al lancio del container. In particolare verrà eseguito il comando *npm start*, che porta Npm ad eseguire gli script nel file package.json. Lì verrà definito uno script per il lancio dell'applicazione. Dalle righe 20 alla 22 si specifica che il container appartiene alla rete dei container dmii e gli viene assegnato l'indirizzo ip 10.0.200.21. Alla riga 23 viene indicato al daemon che il container dovrà essere lanciato solo dopo che il container proxy è partito, in modo da gestire in maniera coordinata l'avvio dei container (altrimenti l'ap-

plicazione potrebbe venire lanciata prima del proxy e non troverebbe il server OpenLDAP al quale si deve connettere)

25-31: Attraverso l'istruzione top-level *networks* vengono definite reti che possono essere riusate da molteplici servizi. In questo caso viene creata una rete *dmii* con alcune configurazioni: alla riga 27 si da un nome alla rete, alla riga 28 con *driver* viene specificato quale driver verrà usato per questa rete. All'ultima riga viene specificato che il set di indirizzi ip che possono essere assegnabili all'interno della rete dmii. Essi sono i 10.0.200.0/24, ovvero saranno disponibili 255 indirizzi ip, cambiando gli ultimi 8 bit dei 32.

4.4 Configurazione del Proxy OpenLDAP

Un altro passo fondamentale nella costruzione del progetto è la configurazione del Server OpenLDAP. La configurazione del server OpenLDAP viene fatta tramite il file di configurazione *slapd.conf* contenuto all'interno della cartella */etc/ldap*. Tramite il comando *slaptest -f /etc/ldap/slapd..conf -F /etc/ldap/slapd.d* questo file di configurazione viene convertito nella configurazione moderna di OpenLDAP, ovvero un insieme di file ldif contenuti all'interno della cartella */etc/ldap/slapd.d*. Il file di configurazione del Server OpenLDAP è:

```

### Schema includes #####
include          /etc/ldap/schema/core.schema
include          /etc/ldap/schema/cosine.schema
include          /etc/ldap/schema/inetorgperson.schema
include          /etc/ldap/schema/misc.schema
include          /etc/ldap/schema/nis.schema
include          /etc/ldap/schema/corba.schema
include          /etc/ldap/schema/duaconf.schema
include          /etc/ldap/schema/dyngroup.schema
include          /etc/ldap/schema/java.schema
include          /etc/ldap/schema/openldap.schema

## Module paths #####
modulepath      /usr/lib/ldap/
moduleload      back_ldap
moduleload      back_ldif
moduleload      back_mdb
moduleload      rwm
moduleload      /usr/lib/ldap/translucent.la
moduleload      /usr/lib/ldap/memberof.la
moduleload      /usr/lib/ldap/pcache.la
moduleload      back_monitor

# Main settings #####
pidfile         /var/run/slapd/slapd.pid
argsfile        /var/run/slapd/slapd.args

database frontend
database monitor

database config
rootdn $ROOT_DN_CONFIG
rootpw $ROOT_PW

```

Figura 4.8: Parte 1 configurazione Server OpenLDAP

Nella prima parte del file [Figura 4.8] di configurazione si indicano quali sono i moduli e le librerie da caricare nel server. In particolare nelle prime righe si trovano i file di schema caricati, che contengono le dichiarazioni degli *object classes*, degli attributi e dei tipi che si vogliono poter utilizzare all'interno della directory LDAP. Successivamente vengono caricati i moduli software che possono essere utilizzati all'interno del server (come ad esempio il backend ldap per poter creare un server proxy, il backend mdb o i moduli che permettono l'utilizzo degli overlay OpendLDAP). Con *pidfile* viene indicato il path del file dove *slapd* memorizzerà il suo PID (process identifier), mentre con *argsfile* il path del file dove gli argomenti utili per l'avvio di *slapd* saranno memorizzati. Le direttive *database frontend*, *database monitor* e *config* servono ad indicare di voler utilizzare questi tipi di database all'interno del server. Il frontend è un database speciale utilizzato per contenere le opzioni a livello di database che dovrebbero essere applicate a tutti gli altri database. Le successive definizioni del database potrebbero anche sovrascrivere alcune impostazioni del frontend. Anche il database di config è speciale. Sia il database di config che quello frontend vengono sempre creati

in modo implicito anche se non sono configurati esplicitamente e vengono creati prima di qualsiasi altro database. Nel database config con *olcrootdn* e *olcrootpasswd* si specifica un particolare tipo di utente che è autorizzato a fare qualsiasi operazione su quel database. Con l'istruzione *database monitor* viene abilitato questo particolare tipo di backend. Quando abilitato, il database monitor dinamicamente genera e ritorna oggetti in risposta a ricerche nel sottoalbero *cn=Monitor*, in cui ogni oggetto contiene informazioni riguardo un particolare aspetto del server OpenLDAP. Nella seconda parte del file di configurazione vengono settate le impostazioni dei database che conterranno i dati delle entries, database tutti di tipo *mdb*. Il backend **Memory-Mapped Database(mdb)** è il backend consigliato per il server *slapd*. Esso supporta indicizzazione, non utilizza caching, è totalmente gerarchico e permette rinomina dei sottoalberi in tempo costante. Verranno creati nel complesso tre database.

```
database mdb
suffix "ou=Gruppi Studenti,dc=unict,dc=ad"
rootdn "CN=monitor,CN=Users,DC=unict,DC=ad"
rootpw $ROOT_PW
index objectClass eq
readonly off
directory /var/lib/ldap
sizelimit 100000
overlay translucent
uri $URI
lastmod off
idassert-bind bindmethod=simple
binddn=$BIND_DN
credentials=$BIND_PW
mode=self
sizelimit 1000000
access to dn.subtree="ou=Gruppi Studenti,dc=unict,dc=ad"
by dn.exact=$BIND_DN write
overlay memberof
subordinate
```

Figura 4.9: Parte 2 configurazione Server OpenLDAP

Il primo database è stato configurato [Figura 4.9] in modo che venga contattato nel caso di query di suffisso *ou=Gruppi Studenti,dc=unict,dc=ad*, ovvero le entries che nell'Active Directory d'ateneo si trovano all'interno della Organizational Unit "Gruppi Studenti". Con la direttiva *suffix* si specifica il suffisso del DN delle query che dovranno essere passate al backend di questo database. Con *rootdn* e *rootpasswd* si indicano il DN e la password che individuano un'entità che non è soggetta a controlli di accessi o a limiti nelle operazioni. La direttiva *index* specifica gli indici da mantenere per il dato attributo. Gli indici che vengono definiti corrispondono ai tipi più comuni di match che dovrebbero essere usati nei filtri di ricerca, in modo da ottimizzare il salvataggio e la risposta alle richieste. La direttiva *readonly off* permette di rendere il database modificabile, invece di averlo in solo lettura.

ra. Con *directory* si esplicita la directory dove verranno salvati i file mdb contenenti i dati salvati nel database. *Sizelimit* specifica il massimo numero di entries che possono essere ritornati da un'operazione di ricerca. La parte fondamentale della configurazione di questo database è l'utilizzo dell'**overlay Translucent**: questo permette al server, nel caso di query con quel suffisso, di ritornare anche le entries che sono presenti nell'Active Directory remoto, potendone modificare localmente anche alcuni attributi prima di ritornarli al client. La direttiva *idassert-bind* serve a specificare un'entità col quale il server si autenticherà al server remoto nel caso di un'autenticazione locale andata a buon fine. Vengono stabiliti poi i controlli di accesso permettendo le scritture e letture sul database solo ad una particolare entry con il dn specificato. Viene utilizzato l'overlay *memberOf* per la gestione dei gruppi. Con *subordinate* si specifica che questo database sarà un "subordinato" a quello successivo. Poichè il successivo database avrà suffisso "dc=unict,dc=ad", questa direttiva permette di specificare che se verrà fatta una ricerca con dominio "dc=unict,dc=ad", dovranno essere ritornati anche i risultati di questo database, di suffisso "ou=Gruppi Studenti,dc=unict,dc=ad".

```

database    mdb
suffix      "ou=Studenti,dc=unict,dc=ad"
rootdn     "CN=Administrator,CN=Users,DC=unict,DC=ad"
rootpw      $ROOT_PW
maxsize    85899345920
index objectClass eq
readonly off
directory  /var/lib/ldap
sizelimit  100000
subordinate

overlay rwm
rwm-map attribute memberOfGroup memberOf

overlay memberof
overlay translucent
translucent_local $LOCAL_ATTRIBUTES
translucent_remote $REMOTE_ATTRIBUTES
uri          $URI
lastmod    off
idassert-bind bindmethod=simple
            binddn=$BIND_DN
            credentials=$BIND_PW
            mode=self
sizelimit 1000000
access to dn.subtree="ou=Studenti,dc=unict,dc=ad"
        by dn.exact=$BIND_DN write
subordinate

```

Figura 4.10: Parte 3 configurazione Server OpenLDAP

La configurazione del secondo database [Figura 4.10] è la parte fondamentale di tutto il progetto. Viene come prima specificato l'uso del database di tipo mdb. Con la direttiva *suffix* si indica il suffisso del DN delle query che dovranno essere passate al backend di questo database. Le direttive successive hanno le stesse funzionalità descritte nella sezione precedente. Con *subordinate* viene indicato che questo database sarà un "subordinato"

a quello successivo. In particolare poichè il successivo database avrà suffisso "dc=unict,dc=ad", questa direttiva permette di specificare che se verrà fatta una ricerca con dominio "dc=unict,dc=ad", dovranno essere ritornati anche i risultati di questo database di suffisso "ou=Studenti,dc=unict,dc=ad".

Viene poi configurato il primo overlay. In particolare l'*overlay rwm* ha come scopo quello di fare semplici mapping di objectClass/attributeType dall'insieme locale ad un insieme esterno definito e viceversa. Attraverso la direttiva *rwm-map* l'overlay mappa valori che provengono da un server esterno a differenti valori nel server locale. In questa configurazione viene indicato che tutti gli attributi di tipo memberOf vengano ritornati a chi li richiede col tipo memberOfGroup. Questa operazione è necessaria per far sì che il valore memberOfGroup possa essere multi-valued, a differenza dell'attributo memberOf del software OpenLDAP.

Successivamente viene definito l'utilizzo di un ulteriore overlay, l'overlay *memberOf*. Questo overlay aggiorna un attributo (di default MemberOf che in questo caso verrà mappato con MemberOfGroup) se quel determinato dn dell'entry è presente nell'attributo member di un'entry di tipo objectClass GrupoOfNames. Questo garantisce la manutenzione di una lista di gruppi alla quale un'entry appartiene, invece che modificare ogni volta un'entry aggiungendo un gruppo alla quale appartiene.

Si esplicita poi l'utilizzo dell'overlay *Translucent*. Questo overlay è quello che da la funzionalità di Proxy all'Active Directory d'ateneo. Grazie all'opzione *uri* viene specificato l'indirizzo ip del server remoto alla quale verranno rigirate le richieste che arrivano a quel backend. Le entries che provengono da un remoto LDAP server possono avere alcuni o tutti gli attributi sovrascritti con valori locali, o alcuni attributi aggiunti, grazie ad entries con lo stesso DN salvate nel database locale. Con Translucent local si specificano una lista di attributi che devono essere ricercati nel database locale se specificati nei filtri di ricerca, mentre con Translucent remote una lista di attributi che devono essere cercati nel database remoto quando usati in un filtro di ricerca. Il backend LDAP (che è utilizzato sotto all'overlay Translucent) supporta da OpenLDAP 2.3 l'***identity-assertion***, che serve a garantire un bind al server remoto del quale si sta facendo proxy con un'identità amministrativa specificata. Nel caso in cui con un server OpenLDAP si fa proxy di un server LDAP remoto, il proxy verrà usato da utenti la cui identità non è presente nel server remoto, e quindi non potrebbero essere autenticati. Poichè gli utenti locali possono autenticarsi localmente, il backend proxy può essere istruito per asserire la loro identità e trasferire quello che hanno richiesto al server remoto usando un'identità amministrativa che è memorizzata nel server remoto e che possiede i privilegi necessari. Con *idassert-bind* si esplicita l'identità col quale si fa bind al server remoto. Vengono infine definite le ACL, access con-

trol list. In particolare l'accesso al sottoalbero *ou=Studenti,dc=unict,dc=ad* è permesso solo a quel determinato DN in scrittura (ed anche in lettura, perchè OpenLDAP gestisce il controllo di accesso in questo modo).

```
database mdb
suffix "dc=unict,dc=ad"
readonly off
rootdn "cn=admin,dc=unict,dc=ad"
rootpw $ROOT_PW
sizelimit 100000
access to dn.subtree="ou=Gruppi Locali,dc=unict,dc=ad"
    by dn.exact=$BIND_DN write
access to dn.subtree="ou=Studenti Locali,dc=unict,dc=ad"
    by dn.exact=$BIND_DN write
access to dn.subtree="dc=unict,dc=ad"
    by dn.exact=$BIND_DN write
overlay memberof
```

Figura 4.11: Parte 3 configurazione Server OpenLDAP

In questa ultima parte [Figura 4.11] di configurazione viene configurato un ulteriore database mdb. Si parla di un semplice database locale che risponderà alle richieste al suffisso ”dc=unict,dc=ad”. Viene definita l'entità amministrativa che non sarà costretta a controlli d'accesso o a restrizioni. Alle righe finali vengono definiti i controlli di accesso. Anche su questo database sarà abilitato l'overlay MemberOf.

4.5 Translucent Proxy Admin: struttura e implementazione

L'applicazione web Trasclucent Proxy Admin di questo progetto è sviluppata in Javascript con l'ausilio dell'ambiente di esecuzione Node.js.

Un'applicazione web è un programma, ospitato da un server remoto, al quale si può accedere da ogni browser in qualsiasi dispositivo, che garantisce servizi e svolge i compiti richiesti da un utente. L'applicazione web sviluppata in questo progetto è un'applicazione web dinamica che segue lo stile architettonale RESTful API e che sfrutta anche i concetti di una single-page web application. Con applicazione web dinamica si indica una pagina web che fornisce dati live basati sulle richieste degli utenti , spesso grazie all'utilizzo di database di backend per memorizzare dati pubblici e privati che vogliono poi poter essere gestiti o mostrati all'utente. Con RESTful API si indica uno stile architettonico per le API di un'applicazione che usano le richieste HTTP per accedere ed usare determinati dati. Un API è un pezzo di codice che il programmatore sviluppa nell'applicazione per far sì che un altro programma possa comunicarvi. Una single-page web application è un'applicazione che

risulta molto simile ad un'applicazione desktop perché non interrompe l'experience durante la navigazione tra una pagina e l'altra, ma riscrive la pagina corrente senza richiederne una nuova al server. In una single page application tutto il codice necessario, come quello HTML, Javascript e CSS, viene scaricato per intero durante il primo caricamento della pagina.

4.5.1 Struttura dell'applicazione

4.5.1.1 Design pattern model-view-controller

L'applicazione web Traslucent Proxy Admin si basa sul design pattern *model-view-controller*.

Il pattern richiede che ognuno di questi deve essere separato in differenti oggetti e moduli nell'implementazione di un'applicazione. In generale:

- Il modello contiene solo i dati dell'applicazione. Esso deve mappare i dati che provengono da database e che devono essere mostrati all'utente. Non contiene logica. Ad esempio nell'applicazione web vengono utilizzati come modelli User, Attribute e Group, che servono per mappare in strutture dati i dati provenienti dal database del proxy OpenLDAP in modo da poterli mostrare tramite interfaccia grafica all'utente

```
class User{
    sn;
    givenName
    cn;
    constructor(sn,givenName,cn){
        this.sn = sn;
        this.givenName = givenName;
        this.cn = cn;
    }
}

class Group{
    cn;
    constructor(cn){
        this.cn=cn;
    }
}

class Attribute{
    type;
    value;
    local;
    constructor(type,value){
        this.type = type;
        this.value = value;
        this.local = false;
    }
}
```

Figura 4.12: Modelli usati nella WebApp

- La view presenta il modello dei dati all'utente. La view conosce come accedere al modello dei dati, ma non conosce cosa questi dati significhino

no o come l'utente può manipolarli. All'interno dell'applicazione web si trovano una view che permette la visualizzazione degli utenti che corrispondono ad un criterio di ricerca (quindi la presentazione del modello User), una view che permette la visualizzazione dei dettagli di un utente o di un gruppo a seguito di una ricerca (quindi la presentazione del modello Attribute), una view che permette la visualizzazione dei gruppi che corrispondono a determinati criteri di ricerca (quindi la presentazione del modello Group), e due view che permettono il salvataggio di nuovi utenti e nuovi gruppi.

- Il controller si trova tra il modello e la view. Esso ascolta gli eventi triggerati dalla vista e esegue le reazioni appropriate a questi eventi. In molti casi la reazione è chiamare una funzione sul modello, per mostrare o modificare qualcosa. Poichè sia la vista che il modello sono connessi da un sistema di eventi e notifiche, il risultato di questa azione si riflette automaticamente nella view

Sin dalla sua comparsa l'architettura Model-View-Controller è stata implementata lato server nelle applicazioni, soprattutto grazie all'avvento di framework di sviluppo basati su di esso. La struttura dell'applicazione è quindi così pensata:

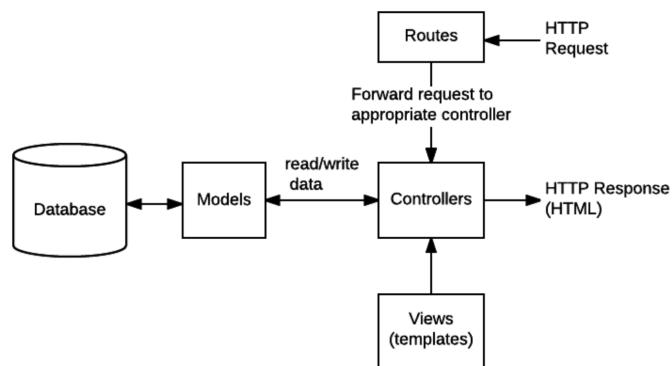


Figura 4.13: Design Pattern Model-View-Controller

4.5.1.2 Architettura three-tier

Con il termine architettura three-tier si intende un'architettura client-server in cui l'interfaccia utente, la logica dell'applicazione e l'archiviazione dei dati con il relativo accesso ad essi sono sviluppati e mantenuti sotto forma di moduli indipendenti. Tipicamente l'interfaccia utente è eseguita sul pc dell'utente che utilizza l'applicazione, la logica risiede in uno o più moduli diversi che costituiscono l'applicazione server e l'archiviazione dati è gestita da un database server. I livelli che costituiscono l'architettura three-tier sono:

- Livello di presentazione: il livello più alto e si può associare al motore di rendering del web browser, che visualizza le informazioni relative ai servizi dell'applicazione (front-end). L'interfaccia utente è formata dall'insieme di web page scritte in HTML, con l'aggiunta di CSS per arricchirne gli stili e Javascript per renderle più dinamiche.
- Livello di applicazione: questo livello è associato al motore applicativo (back-end) che risiede nell'application server ed è costituito dal codice sorgente scritto utilizzando una qualche tecnologia per lo sviluppo di contenuti web dinamici (ex: Javascript e Node.js). E' questo il tier che rende diversa una web app nei confronti di un normale sito web.
- Livello dei dati: questo livello è associato al server che gestisce il database. Le richieste di accesso al database, sia in lettura che scrittura, sono ricevute dal livello di applicazione

Quindi il web browser del client utilizza il protocollo HTTP/HTTPS per inviare richieste al backend. Questo interpreta ciò che è contenuto nella richiesta, esegue eventuali elaborazioni e a sua volta genera altre richieste che vengono destinate al motore del database. Non appena il livello intermedio riceve i dati dal database, genera il risultato in output che sarà interpretato dal browser e restituito all'utente nella forma di una pagina web. Si può pensare anche di introdurre un livello di integrazione al di sopra del livello dei dati che fornisce delle API al livello di applicazione. Queste API espongono l'insieme dei metodi che possono essere utilizzati per gestire i dati del database .

4.5.1.3 Il concetto di API RESTful

Un API REST, è un'interfaccia di programmazione delle applicazioni conforme ai vincoli dello stile architetturale **REST**, che consente l'interazione con servizi web RESTful. Il termine REST è l'acronimo di **Representational State Transfer**.

Con API, acronimo di Application Programming Interface, indichiamo un insieme di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi. L'API stabilisce il contenuto richiesto dal consumatore (quindi il tipo di chiamata) e il contenuto richiesto dal produttore (la risposta). Se si desidera interagire con un computer o un sistema per recuperare informazioni o eseguire una funzione, un'API facilita la comunicazione con il sistema che può così comprendere e soddisfare la richiesta. Un API funge quindi da elemento di intermediazione tra gli utenti o i clienti e le risorse o servizi web che questi intendono ottenere. E' anche il mezzo con il quale un'organizzazione può condividere risorse e informazioni assicurando al tempo stesso sicurezza, controllo e autenticazione, in quanto stabilisce anche i controlli di accesso. Con REST indichiamo un insieme di vincoli architetturali, non un protocollo né uno standard. Quando una richiesta client viene inviata tramite un'API RESTful, questa trasferisce al richiedente o all'endpoint uno stato rappresentativo della risorsa. L'informazione viene consegnata tramite HTTP in uno dei diversi formati: JSON, HTML, Python, testo semplice etc...

Affinchè un API sia considerata RESTful deve rispettare i criteri indicati:

- Un'architettura client-server composta da client, server e risorse con richieste gestite tramite HTTP.
- Una comunicazione client-server stateless, che quindi non prevede la memorizzazione delle informazioni del client tra le varie richieste Get. Tutte le informazioni necessarie al server per restituire i dati devono essere incluse in ogni richiesta.
- Dati memorizzabili nella cache che ottimizzano le interazioni client-server.
- Un'interfaccia uniforme per i componenti, in modo che le informazioni vengano trasferite in una forma standard. Questo è possibile grazie all'utilizzo delle URI (Unique Resource Identifier).
- Un sistema su più livelli che organizza ogni tipo di servizio (ad esempio quelli responsabili per la sicurezza, del bilanciamento del carico ...) che si occupa di recuperare le informazioni richieste in gerarchie, invisibile al client.

Sebbene l'API REST debba essere conforme a questi criteri, il suo impiego è comunque più semplice rispetto a quello del protocollo SOAP, che presenta requisiti più specifici.

4.5.2 Codice dell'applicazione

4.5.2.1 Main dell'applicazione

Si tratta del file che verrà passato a Node per il lancio dell'applicazione. In esso si vanno a richiedere i pacchetti che verranno utilizzati dall'applicazione e a settare alcune configurazioni di oggetti che verranno utilizzati, oltre che a lanciare il server Express in ascolto. In pratica si tratta di poche linee di codice così definite:

```
app_node > appjs > js main.js > ...
1 const express = require("express")
2
3 const app = express();
4
5
6 app.set('view engine', 'ejs');      //setto l'engine ejs
7 app.set('views', process.cwd() + '/views');
8 app.use('/', express.static('./views'));
9 app.use('/', require('./routes/routes'));
10
11 app.listen(8082)
12
```

Figura 4.14: Main dell'applicazione

Nella prima riga si va a richiedere la classe *express* contenuta nel pacchetto *express*, installato all'interno dell'applicazione. Grazie ad esso è possibile istanziare il server. Con la funzione *set* si assegna un nome ad un valore, per poi poter essere utilizzato nella configurazione del server. Con la funzione *listen* viene settata la porta di ascolto del server. Qualsiasi pacchetto che arriva alla porta 8082 del container verrà processato dall'applicazione.

4.5.2.2 Routes dell'applicazione

Una *route* ("rotta") è una sezione di codice Express associata ad un verbo HTTP (GET, POST, PUT, DELETE etc...), ad un path URL, e ad una funzione che è chiamata per gestire quel path. E' possibile creare delle rotte attraverso il middleware *express.Router* che permette di raggruppare le rotte di un particolare sito ed accedere ad esse utilizzando un prefisso comune. Nel file *routes.js* dell'applicazione si vanno a definire le "rotte" dell'applicazione, ovvero la combinazione verbo-url con una specifica funzione del controller che ritornerà poi un risultato al richiedente.

```

1  const express = require("express")
2  const router = express.Router();
3  const { modifyUserView,
4         indexView,
5         loginView,
6         introView,
7         userSectionView,
8         groupSectionView,
9         searchView,
10        userInfoView,
11        modifyUserInfoView,
12        searchGroupsView,
13        groupInfoView,
14        saveGroupView,
15        addMemberView,
16        modifyMembershipView,
17        saveUserView,
18        newUserView,
19        newGroupView,
20        deleteGroupView } = require('../controllers/controller');
21  const ldap = require("ldapjs")
22  const bodyParser = require('body-parser');
23  const session = require('express-session');
24
25
26 // Use sessions in the app
27 router.use(session({
28   name: "LdapCookie",
29   secret : '1234567890abcdefghijklmnopqrstuvwxyz',
30   resave : false,
31   saveUninitialized : true,
32   cookie : { secure : false }
33 }));
34
35 router.use(express.json())
36 router.use(bodyParser.json())
37
38 router.get("/",indexView)
39 router.get("/modifyUser",modifyUserView)
40 router.post("/login",loginView)
41 router.get("/intro",introView)
42 router.get("/userSection",userSectionView)
43 router.get("/groupSection",groupSectionView)
44 router.post("/addMember",addMemberView);
45 router.get("/newUser",newUserView)
46 router.post("/saveUser",saveUserView)
47 router.get("/newGroup",newGroupView)
48 router.get("/search", searchView);
49 router.get("/searchUser", userInfoView)
50 router.post("/modifyUserAttributes", modifyUserInfoView);
51 router.get("/searchGroups", searchGroupsView);
52 router.get("/searchGroup", groupInfoView)
53 router.post("/saveGroup", saveGroupView)
54 router.post("/modifyMembership", modifyMembershipView)
55 router.post("/deleteGroup", deleteGroupView)
56
57
58 module.exports = router;
59

```

Figura 4.15: Routes dell'applicazione

Nelle prime righe si vanno ad importare, oltre alla classe Express, quelle funzioni che sono state definite nel controller e che andranno a gestire i parametri in arrivo dalle richieste generando risorse che verranno mandate in

risposta al client. Alle righe 19 e 20 vengono importati i moduli che saranno utilizzati per usufruire di alcune funzioni come middleware. Utilizziamo il middleware *session* che permette di gestire sessioni di richieste degli utenti automaticamente in base alle opzioni definite, e i middleware *express.json* e *body-parser* che permettono la gestione del body delle richieste che arrivano trasformandoli in oggetti JSON più facilmente gestibili. Questo sarà fondamentale nell'applicazione, dove il client tramite dei form invierà informazioni su nuovi utenti , nuovi attributi etc... che il controller gestirà facilmente come oggetti JSON.

Dalle righe 33 a 47 viene definito il trittico verbo-uri-funzione che permetterà l'invocazione della funzione definita nel controller all'arrivo di una richiesta con un determinato verbo ad un determinato URI (Unique resource identifier). Ad esempio all'arrivo di una richiesta GET di tipo

`http://applicazioneweb/saveUser` verrà invocata la funzione `saveUserView` implementata nel controller che procederà al salvataggio di un nuovo utente all'interno del server OpenLDAP.

Alla riga 49 viene utilizzato l'oggetto *module* di Node che permette di esportare oggetti, attraverso la proprietà *exports*, che possono essere importati ed utilizzati in altre parti dell'applicazione. In questo caso viene esportato l'oggetto `Router`, utilizzato all'interno del controller.

4.5.2.3 Controller dell'applicazione

Nel file `controller.js` vengono definite le funzioni del controller che gestiranno le richieste alle API dell'applicazione. Qualsiasi operazione CRUD (CREATE, READ, UPDATE, DELETE) sui dati del Server OpenLDAP avrà una funzione definita all'interno del controller. All'interno di queste funzioni vengono gestiti i dati che arrivano dalle richieste e i dati che devono essere mandati nelle risposte grazie alle classi `Request` e `Response` del pacchetto `express`. Verranno utilizzate qui le funzioni e gli oggetti del pacchetto `ldapjs` per poter comunicare con il server OpenLDAP per la memorizzare di nuove informazioni nella directory del proxy. Si farà questo grazie alla definizione di un oggetto `client` dalla classe `LdapClient` nel quale setteremo un URL che corrisponde all'indirizzo ip del container che contiene il server OpenLDAP.

```

23 |
24   const client = new LdapClient({
25     url: serverUrl
26   }
27

```

Figura 4.16: Client Ldapjs

All'interno di questo file vengono definite le varie funzioni che verranno richiamate all'arrivo di una determinata richiesta, processeranno i parametri arrivati e genereranno una risposta che verrà inviata al client sotto forma di pagina HTML (tramite l'engine EJS) o tramite oggetto JSON o risposta testuale. Le funzioni implementate sono:

- *loginView*: questa funzione prende il body della richiesta POST che contiene nome utente e password e tenta, tramite il client pre-definito, di fare un bind con l'OpenLDAP server Proxy. Il Proxy girerà la richiesta di autenticazione all'Active Directory. In caso di risposta positiva verrà inizializzata una sessione per quell'utente e verrà mandato una risposta testuale "loginOk". In caso di risposta negativa verrà inviato invece uno status Error 401 e un messaggio di errore.
- *indexView*: questa funzione verrà invocata quando il client richiede la root dell'applicazione web. Il server a questa richiesta risponderà inviando la pagina HTML che contiene il form di login.
- *introView*: questa funzione verrà invocata quando il client, dopo un login con successo, richiede la root dell'applicazione. In questo caso il server si occuperà solo di mandare la view index, ovvero la pagina intro.html che verrà reindirizzata dal browser.
- *userSectionView* e *groupSectionView*: queste funzioni verranno invocate quando il client, al click su uno dei due bottoni presenti nella schermata iniziale, selezionerà se voler entrare nella sezione per la gestione di un singolo utente o di un gruppo. Essa si occuperà semplicemente di ritornare al client la view corrispondente (userSection o groupSection).
- *searchView*: questa funzione viene invocata quando un utente fa una richiesta di ricerca degli utenti con particolari caratteristiche (nome, cognome etc..). L'applicazione utilizzerà la funzione *search* del client di ldapjs con particolari opzioni in base ai parametri della richiesta per ottenere dall'OpenLDAP Proxy gli utenti che corrispondono ai parametri di ricerca. Ottenuto il risultato, manderà un insieme di oggetti json in cui ogni oggetto corrisponde ad un User con un givenName, un CN, un Surname ed una Mail.
- *userInfoView*: questa funzione viene invocata quando un utente vuole ottenere informazioni dettagliate su una particolare entità. Il client invia una richiesta GET con parametro il CN dell'utente che si vuole approfondire. Il server farà una richiesta all'OpenLDAP Proxy per

ottenere le informazioni su quell’entità. Un particolare dettaglio viene implementato in questa funzione: poichè una delle funzionalità che permette l’OpenLDAP Server insieme all’overlay Translucent è la possibilità di fare override degli attributi remoti, l’applicazione fa una richiesta parallela alla directory d’ateneo. Confronta poi gli attributi ritornati dalle due query, settando un attributo del modello Attribute nel caso in cui quell’attributo risulta modificato in locale. L’applicazione ritorna al client poi un insieme di oggetti Attribute che verranno gestiti dal codice lato client legato alla pagina userSection.

- *modifyUserView*: questa funzione viene invocata alla richiesta di un utente di modificare le informazioni di una determinata entità. In questo caso l’applicazione fa una richiesta al proxy delle informazioni dettagliate per quell’utente. In questo caso si fa uso dell’engine EJS. Il server ritorna la view modifyUser passando come valori per valorizzare le varibili template inserite i dati dell’utente che si vuole modificare.
- *modifyUserInfoView*: questa funzione viene invocata quando il client invia tramite una richiesta post i valori che vogliono essere sostituiti ai vecchi valori. In questo caso l’applicazione utilizzerà la funzione *modify* del client per modificare gli attributi dell’utente con quel particolare Distinguished Name. In caso di successo verrà mandata una risposta testuale positiva al client che cogliendolo mostrerà un messaggio di alert, mentre in caso negativo succederà la stessa cosa ma con un alert negativo.
- *searchGroupView*: questa funzione viene invocata quando un utente fa una richiesta di ricerca di gruppi con particolari caratteristiche (nome...). L’applicazione utilizzerà la funzione *search* del client di ldapjs con particolari opzioni in base ai parametri della richiesta per ottenere dall’OpenLDAP Proxy i gruppi che corrispondono ai parametri di ricerca. Ottenuto il risultato, manderà un insieme di oggetti json in cui ogni oggetto corrisponde ad un modello Group.
- *groupInfoView*: questa funzione viene invocata quando un utente vuole ottenere informazioni dettagliate su una particolare entità group. In particolare il client invia una richiesta GET con parametro il CN del gruppo che si vuole approfondire. Il server, come prima, farà una richiesta all’openLDAP proxy per ottenere le informazioi su quell’entità. L’applicazione ritorna al client poi un insieme di oggetti Attribute che verranno gestiti dal codice lato client legato alla pagina userSection.

- *saveGroupview*: questa funzione viene invocata quando un utente fa richiesta di salvataggio di un nuovo gruppo. Verrà utilizzata la funzione *add* del client di *ldapjs-client* per salvare un'entità con quel dn e quegli attributi. Nel caso di successo verrà invocata una risposta testuale positiva, altrimenti negativa.
- *addMemberView*: questa funzione viene invocata quando un utente vuole poter aggiungere un utente all'interno di un Gruppo Locale. In questo caso verrà aggiunto l'attributo *member* al gruppo con quel particolare DN. Grazie all'overlay *memberOf*, alla ricerca dell'utente con quel particolare dn spunterà anche l'attributo *memberOf* inizializzato a quel gruppo. In questa funzione verrà inserito all'interno di quell'entità anche un attributo *memberOf* per facilitare la gestione e rimozione della membership.
- *modifyMembership View*: questa funzione viene invocata quando un utente richiede la rimozione di un membro da un gruppo. Da quel gruppo verrà eliminato l'attributo *member* con valore quel particolare dn, e verrà anche rimosso l'attributo *memberOf* di quell'utente inizializzato con il dn del gruppo.
- *saveUserView*: questa funzione viene invocata quando un utente richiede il salvataggio di un utente Locale. In questo caso il server riceve gli attributi dell'utente da salvare e tramite la funzione *add* del client di *ldapjs-client* procederà alla richiesta di salvataggio di una nuova entità all'OpenLDAP proxy.
- *deleteGroup*: questa funzione viene invocata quando un utente richiede la rimozione di un determinato gruppo locale.

4.5.3 Views dell'applicazione

Le views dell'applicazione sono semplicemente le pagine web mostrate all'utente. Esse sono pagine che mostrano il testo, le immagini di un sito web, link etc... In questa applicazione in alcuni casi sono stati utilizzati template che il controller andrà poi a riempire con alcuni dati in base alla richiesta arrivata. Nell'applicazione web si hanno 7 views, ognuna che mostra all'utente una semplice interfaccia utilizzabile per richiedere servizi all'applicazione:

- *index* questa è la view che verrà ritornata dal server alla richiesta della root da parte del browser. Essa è l'interfaccia di login all'applicazione. E' costruita in maniera semplice tramite un form di login nel quale

inserire il codice fiscale e la password di autenticazione ad Active Directory. Il click sul pulsante e la gestione della risposta alla richiesta di login verrà gestito tramite uno script Javascript index.js.

- *intro* in questa view vengono mostrati semplicemente due bottoni, uno per l'ingresso alla sezione di gestione dell'utente ed uno per l'ingresso alla sezione di gestione dei gruppi.
 - *userSection* in questa view si mostrano semplicemente i vari campi di input per la costruzione di un filtro per la ricerca di un determinato utente. Una volta costruito il filtro, si può fare una richiesta di ricerca all'applicazione web che la rigirerà al server OpenLDAP. All'arrivo dei risultati verranno mostrate tutte le entries che corrispondono ai parametri di ricerca. In questa view emerge il concetto di Single-page-Application. La visualizzazione delle informazioni dettagliate di un utente avviene infatti tramite un'ulteriore richiesta al server. All'arrivo della risposta, verrà nascosta la sezione contenente le entries che corrispondevano ai parametri di ricerca e verrà creata una nuova sezione contenente le info dell'utente che si vuole analizzare. Questa sezione conterrà pure un bottone per la modifica degli attributi dell'utente ed un bottone per l'aggiunta dell'utente ad un gruppo locale.
 - *modifyUser* in questa view emerge l'utilizzo dell'engine EJS. Viene costruito infatti un template di view, dove alcuni campi verranno valorizzati dal controller. Ad esempio la seguente view tratta una variabile *entries*: che verrà poi popolata dal controller in base alla richiesta:

```
<h1 style="text-align: center"> PAGINA INFORMAZIONI UTENTE </h1>
<% if(entries.length > 0) { %>

<div id="userInfo">
  <% for(let i = 0; i < entries.length ; i++) { %>

    <% if(entries[i].type == "MEMBEROFGROUP")%>
      <div class="entryRow" style="display: none;"> <span><=%=entries[i].value.name%></span>
        <% for(var j = 0; j < entries[i].value.length; j++) { %>
          <div class="entryRow"> <span><=%=entries[i].type%></span> &nbs
        <% } %>
    <% } %>
  <% } %>
</div>
```

Figura 4.17: ModifyUser view template

```
res.render("./modifyUser",{entries: results});
```

Figura 4.18: ModifyUser valorizzazione del template

Questo approccio permette la creazione di un template di view che sarà uguale per qualsiasi utente che si vuole analizzare, ma che avrà determinati campi customizzabili in base all'entità attenzionata.

- *groupSection* questa view permette la gestione dei gruppi. Vengono mostrati i vari campi per creare il filtro di ricerca di determinati gruppi. Per ogni gruppo si possono visualizzare le informazioni dettagliate (ad esempio i member).
- *newUser* e *newGroup* sono le view che verranno tornate alle richieste di creazione di nuovi gruppi o utenti locali

Il layout di qualsiasi view è creato grazie all'utilizzo di regole CSS vanilla e grazie all'utilizzo di alcuni framework frontend per customizzarle in maniera moderna.

4.6 Requisiti dell'applicazione

La configurazione di un **OpenLDAP Translucent Proxy** all'Active Directory d'ateneo vuole permettere una gestione "locale" degli utenti di un dipartimento, permettendo la creazione di gruppi locali (come può essere ad esempio un gruppo "Test") e l'inserimento in essi di determinate entità, oppure l'override degli attributi di un'entità proveniente dal servizio di directory d'ateneo. D'altra parte l'applicazione web ha come unico scopo quello di fornire un'interfaccia web user-friendly per favorire agli amministratori l'utilizzo delle funzionalità offerte dall'**OpenLDAP Translucent Proxy**.

4.6.1 Funzionalità previste dall'**OpenLDAP Translucent Proxy**

La configurazione dell'**OpenLDAP Translucent Proxy** deve permettere:

- la creazione di un gruppo locale al quale potrà poi essere aggiunto un utente esistente nell'Active Directory d'ateneo o un utente fittizio creato localmente.
- la creazione di un utente fittizio con particolari attributi.
- l'override di determinati attributi provenienti dall'Active Directory d'ateneo. Un amministratore potrebbe quindi cambiare localmente un attributo (ad esempio il Cognome) di un'entry proveniente dall'Active Directory.
- ad un amministratore, in fase di configurazione, di decidere quali attributi specificati in un filtro di ricerca cercare all'interno dell'OpenLDAP proxy e quali invece cercare all'interno dell'Active Directory d'ateneo.

- ad un amministratore di creare localmente nuove Unità Organizzative, nuovi gruppi e nuove entità sempre sotto il dominio $dc=unict, dc=ad$, creando quindi un'estensione dell'insieme di entries remote.

4.6.2 Funzionalità previste dall'applicazione web

L'applicazione web deve facilitare la gestione di tutti gli scopi pre-citati. Deve quindi permettere:

- ad un particolare utente di autenticarsi tramite una pagina di Login. Le credenziali che l'utente inserirà saranno le credenziali con la quale l'utente è registrato all'Active Directory d'ateneo. In base alle credenziali inserite l'applicazione permetterà determinate operazioni e ne vieterà altre
- ad un utente di ricercare entità che corrispondono a determinati parametri di ricerca
- ad un utente di visualizzare informazioni dettagliate riguardo un particolare utente selezionato
- la modifica di particolari attributi locali di un'entità selezionata
- la creazione di gruppi ed utenti locali
- la ricerca di gruppi che corrispondono a determinati parametri di ricerca e la visualizzazione delle informazioni dettagliate (così come i member...) di un gruppo selezionato
- l'aggiunta o la rimozione di un utente da un particolare gruppo
- un menù che permetta la navigazione all'interno dell'applicazione web ed il logout

4.7 Esempio utilizzo dell'applicazione

L'applicazione vuole essere facile da utilizzare in modo da semplificare le operazioni richieste all'OpenLDAP Server per la gestione delle utenze.

L'applicazione si presenta con una schermata di Login dove un utente può inserire le credenziali:

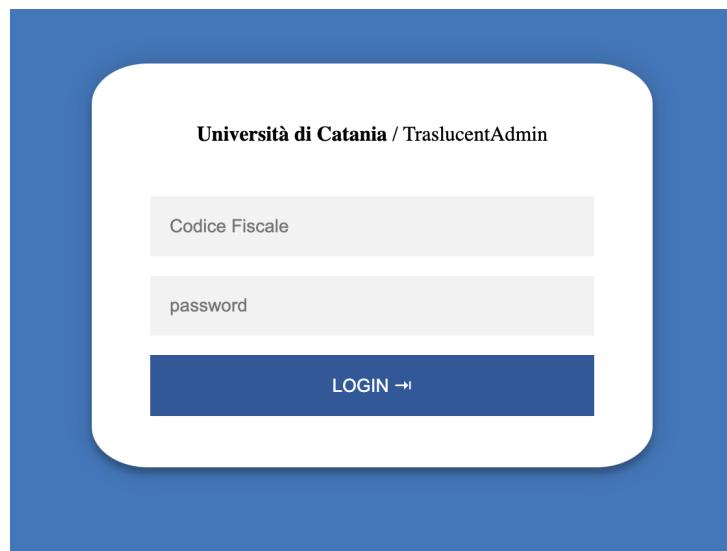


Figura 4.19: Login

Nel caso di login con successo, all'utente si presenterà una schermata che permetterà la scelta di utilizzo della sezione di gestione degli utenti o la sezione di gestione dei gruppi:



Figura 4.20: Intro

Se l'utente sceglie di entrare nella sezione di gestione degli utenti, si presenterà una schermata che permette l'inserimento di vari campi che andranno a formare il filtro di ricerca. Una volta avviata la ricerca, si presenteranno all'utente tutte le entries corrispondenti ai parametri.

Figura 4.21: Sezione Gestione Utenti

Ad esempio alla ricerca di tutti gli utenti corrispondenti al cognome Didomenico, si presenteranno all'utente le entries:

Utente	Codice Fiscale	Email	
LAVINIA DIDOMENICO	DEMAU19880911E00	lavinia.didomenico@unict.it	
GIUSEPPINA DIDOMENICO	DEMAU19880911E00P	giuseppina.didomenico@unict.it	
NICOLA DIDOMENICO	DEMAU19880911E00H	nicola.didomenico@unict.it	
CARMELA DIDOMENICO	DEMAU19880911E00G	carmela.didomenico@unict.it	
MARTINA DIDOMENICO	DEMAU19880911E00P	martina.didomenico@unict.it	
SIMONE DIDOMENICO	DEMAU19880911E00F	simone.didomenico@unict.it	
MARTA DIDOMENICO	DEMAU19880911E00F	marta.didomenico@unict.it	
PAOLO DIDOMENICO	DEMAU19880911E00T	paolo.didomenico@unict.it	

Figura 4.22: Utenti cercati

Una volta mostrate le utenze che presentano determinate caratteristiche, l'utente può selezionare un'entry per visualizzarne le informazioni dettagliate o modificarle.

Per ogni entry verranno visualizzate sia le informazioni salvate localmente sia quelle provenienti dall'Active Directory. Nella scheda dell'utente cliccando

sulla riga "Altri attributi" verranno mostrati gli attributi meno importanti per quell'account.



Figura 4.23: Info Utente

Nella figura 4.23 troviamo le informazioni riguardanti l'entità Lorenzo Dido-menico, che è stato aggiunto al gruppo locale "Test".

Un utente può anche modificare gli attributi di una determinata entry.

Si procederà col fare un overlay degli attributi locali nell'OpenLDAP Trans-lucent Proxy. Un'indicazione all'utente che un attributo è modificato local-mente è fornito con un colore differente nella visualizzazione.

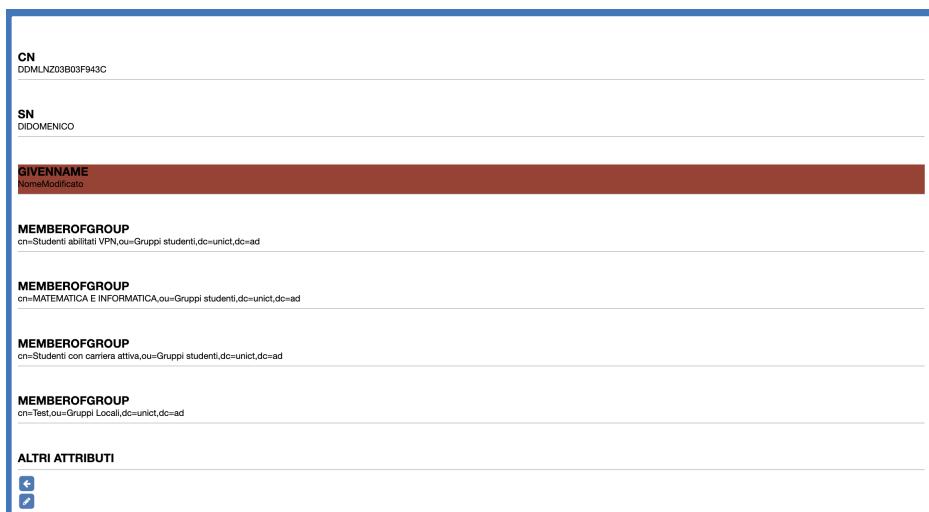


Figura 4.24: Attributo Utente modificato localmente

Nella figura 4.24 vediamo come si presenta la scheda dell’utente Lorenzo Domenico dopo che il nome è stato modificato in ”NomeModificato”.

Nel caso in cui l’utente vuole utilizzare la sezione di gestione dei gruppi, si presenterà una schermata che permetterà la ricerca di gruppi con particolari caratteristiche, oltre alla possibilità di creazione di gruppi o utenti locali:

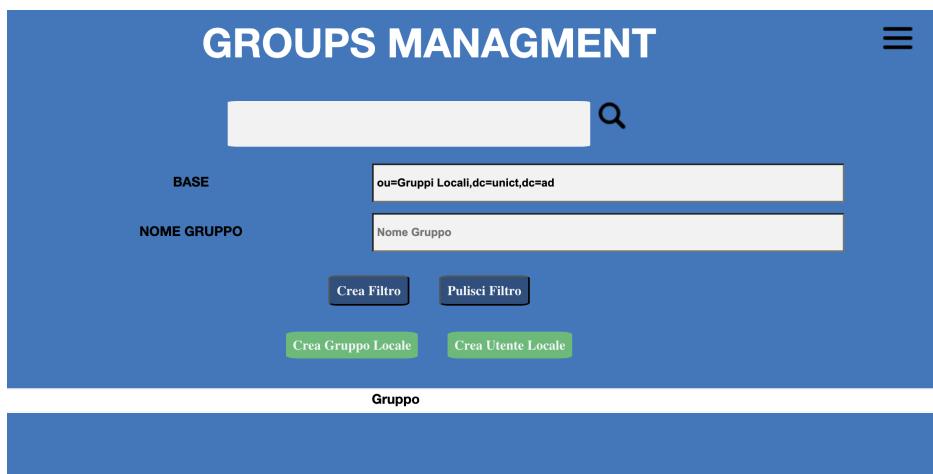


Figura 4.25: Sezione gestione gruppi

L’utente avrà la possibilità di creare un gruppo locale, settandone il nome ed eventuali membri:



Figura 4.26: Sezione nuovo gruppo

Un utente potrà creare un gruppo presente solo nel database locale ed aggiungervi un'entità remota:

Figura 4.27: Info Gruppo

In questo esempio è stato creato un gruppo "Test" ed è stato messo come membro quell'utente (specificandone il DN).

L'applicazione permette anche la creazione di un nuovo utente esistente solo nel database locale, attraverso una semplice interfaccia:

Figura 4.28: Sezione nuovo Utente

Queste sono le principali funzionalità messe a disposizione dall'applicazione Translucent Proxy Admin. L'applicazione web presenta anche un menu per facilitare la navigazione tra le varie sezioni all'utente. Un utente, dopo aver eseguito le operazioni, potrà semplicemente uscire dall'applicazione tramite un bottone di Logout.

Conclusione

L'intento finale di questa tesi era la presentazione di un progetto che potesse essere utile all'Università di Catania per migliorare la gestione delle utenze. Il progetto si pone come obiettivo quello di proporre una gestione "locale" degli utenti dell'Università, le cui informazioni sono memorizzate in Active Directory. Il progetto infatti vuole creare un nuovo livello di gestione al di sopra delle informazioni originali in modo da poterle modificare in base a delle esigenze particolari senza compromettere la struttura e i dati originari.

Nella tesi inizialmente si parla di quelli che sono gli standard odierni per la gestione degli utenti e delle risorse di un'azienda, che permettono la memorizzazione delle informazioni su qualsiasi oggetto di un'organizzazione e la gestione delle autorizzazioni. Gli strumenti ed i protocolli descritti nella tesi sono odiernamente utilizzati per raggiungere questi scopi (ad esempio Active Directory).

Per raggiungere gli obiettivi finali del progetto è stato necessario uno studio approfondito del protocollo LDAP e della sua più famosa implementazione, OpenLDAP. All'interno della tesi si descrivono quindi alcune delle caratteristiche principali di OpenLDAP e di come esso è stato utilizzato e configurato all'interno del progetto. Il progetto prevede anche l'implementazione di un'applicazione web che fornisca all'utente un'interfaccia semplice con la quale interagire col servizio di directory. Per questo all'interno della relazione si descrivono quali sono le tecnologie usate oggi all'interno del mondo del web e di come esse sono state utilizzate all'interno del progetto.

Il progetto potrebbe essere utile a tutte quelle aziende che vogliono rendere più granulare la gestione di utenti e risorse, abbandonano una gestione totale di tutte le componenti organizzative e indirizzandola verso una gestione settoriale o dipartimentale.

Il progetto in futuro potrebbe essere ulteriormente migliorato. Potrebbe ad

esempio essere aggiunto all'interno dell'applicazione la possibilità di accoppiare ad una membership utente-gruppo una scadenza della stessa, creando un processo backend che ad intervalli di tempo vada a controllare la "expiration date" della membership e quindi andarla ad eliminare. Un banale utilizzo in futuro di questa funzionalità potrebbe essere: supponiamo che nasca l'esigenza di creare all'interno di un dipartimento la creazione di un gruppo locale "Esame di ..." e garantire determinati privilegi solo a quelli studenti che appartengono a questo gruppo locale. La membership a questo gruppo dovrebbe rimanere valida fino alla data dell'esame, per poi essere eliminata. Un amministratore potrebbe essere svincolato dall'eliminare tale membership grazie alla routine di pulizia che andrebbe ad eliminarla automaticamente.

Le funzionalità sviluppate nell'applicazione e quelle future potrebbero quindi in futuro facilitare la gestione aziendale delle risorse e delle utenze.

Bibliografia

- [1] CyberArk , *What is a Directory Service* , <https://www.cyberark.com/it/what-is/directory-services/>.
- [2] Microsoft Learn, *The LDAP directory service model*, <https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ldap/the-ldap-directory-service-model>
- [3] Techiepraveen, *Basic Active Directory Components*, <https://techiepraveen.wordpress.com/2010/09/04/basic-active-directory-components/>
- [4] Geeksforgeeks, *Introduction of Active Directory Domain Services*, <https://www.geeksforgeeks.org/introduction-of-active-directory-domain-services/>
- [5] Proofpoint, *What is Active Directory*, <https://www.proofpoint.com/us/threat-reference/active-directory>
- [6] Jim Sermersheim, RFC 4511, *Lightweight Directory Access Protocol (LDAP): The Protocol* <https://www.rfc-editor.org/rfc/rfc4511>
- [7] RedHat, *Cos'è l'Autenticazione LDAP (Lightweight Directory Access Protocol)?*,<https://www.redhat.com/it/topics/security/what-is-ldap-authentication>
- [8] IONOS, *LDAP: definizione e spiegazione*, <https://www.ionos.it/digitalguide/server/know-how/ldap/>
- [9] LDAP, *Learn About Ldap*, <https://ldap.com/learn-about-ldap/>
- [10] University of Michigan, *OpenLDAP Software 2.6 Administrator's Guide*
- [11] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne *Sistemi Operativi, Concetti ed esempi*

- [12] IBM, *Che cosa sono i Container?*,<https://www.ibm.com/it-it/topics/containers>
- [13] Bootdey, <https://www.bootdey.com/>