UNIVERSITÀ
DI PARMA

DIPARTIMENTO di INGEGNERIA e ARCHITETTURA

INGEGNERIA DEI SISTEMI INFORMATIVI
RETI DI TELECOMUNICAZIONI

UNIPR MESSENGER

Progetto realizzato da:
Giuseppe Urbano
Inderpreet Singh

Anno accademico 2018/2019

L’idea di progetto

L’idea che sta alla base di UniprMessenger è quella di creare una semplice piattaforma lato client di instant messaging.
La piattaforma sfrutta il protocollo XMPP (Extensible Messaging and Presence Protocol) per la comunicazione in tempo reale tra gli utenti ed è costituita da un solo dominio (messenger.unipr.it) il quale risiede su un server locale openfire.

Configurazione Openfire

Openfire è un server per instant messaging e chat di gruppo per il protocollo XMPP, sviluppato dalla Ignite Realtime in Java e concesso in licenza con Apache License 2.0.
E’ multipiattaforma (Linux, MacOS, Windows) e fornisce un’interfaccia chiara e semplice per configurare il proprio server XMPP in locale.

Download Openfire: https://www.igniterealtime.org/downloads/index.jsp#openfire

Una volta installato il software per il proprio sistema operativo, si può procedere alla configurazione riportata qui di seguito:

1) Digitare “localhost:9090” in un browser qualsiasi per aprire l’interfaccia di configurazione di Openfire e seguire la procedura.
2) In **Server Settings** impostare il dominio “unipr.messenger.it” e lasciare predefinite le porte già presenti.
3) In **Database Settings** scegliere l’opzione Embedded Database.
4) In **Administrator Account** lasciare l’email predefinita “admin@example.com” e impostare una password a propria scelta.

Una volta completata la procedura di configurazione aprire il server digitando sul browser “localhost:9090”. A questo punto effettuare il login con utente “admin” e utilizzare la password scelta precedentemente. Nella sezione UsersGroup è possibile aggiungere nuovi utenti direttamente dall’interfaccia Openfire.

Gli utenti registrati compariranno nella sezione User Summary ed è possibile controllare in tempo reali quali fra questi è online.

Il protocollo XMPP

Il protocollo XMPP (Extensible Messaging and Presence Protocol), viene sviluppato dall'esigenza di un protocollo di comunicazione in tempo reale aperto, sicuro e decentralizzato, alternativo ai servizi di instant messaging.
XMPP offre diversi vantaggi rispetto ai protocolli proprietari concorrenti:

- **Apertura** → il protocollo è aperto, libero, gratuito e pubblico. Chiunque può scrivere la propria implementazione, sia lato server sia lato client, e produrre ulteriori librerie di supporto. Questo fattore è stato determinante per l'implementazione di tale protocollo nella nostra piattaforma.

- **Standardizzazione** → il protocollo è uno standard IETF basato su stream XML. Gli RFC di riferimento sono RFC 3920 (core) e RFC 3921 (estensioni al protocollo), RFC 3922 (mappatura di XMPP su Common Presence and Instant Messaging, CPIM) e RFC 3923 (cifratura end-to-end).

- **Decentralizzazione** → grazie all'architettura del protocollo ogni persona può realizzare e/o avviare un proprio server XMPP, mantenendo al contempo la possibilità di intercomunicare con gli altri server della rete.

- **Sicurezza** → i server possono essere isolati dalla rete pubblica, inoltre è possibile cifrare le connessioni grazie alle tecnologie SASL e TLS.

Architettura di XMPP

I server XMPP costituiscono una rete simile a quella del servizio mail. Ogni server, se configurato opportunamente, può inoltrare messaggi diretti ad utenze non sue.
Questo significa che due utenti registrati su domini diversi possono comunicare tra di loro, purché i rispettivi server si conoscano.
Per questioni di praticità in UniprMessenger la rete è costituita da un unico server Openfire nel quale è configurato un solo dominio (@unipr.messenger.it), per cui non ci sono comunicazioni tra server e server, ma solo tra client e server.
Le comunicazioni avvengono sulla porta 5222.

Addresses, Dominio e Risorse

Ogni entità XMPP ha bisogno di un indirizzo chiamato JabberID.
I JabberID per gli utenti sono simili agli indirizzi email (user@unipr.messenger.it)

Ogni JabberID appartiene ad un unico dominio.

Ogni volta che un client si collega a un server XMPP viene scelto, dal client o dal server, un identificatore di risorsa per quella connessione. Questa risorsa viene aggiunta al nostro indirizzo. Le risorse sono case sensitive.

Nella nostra implementazione la risorsa viene generata random dal server.

$Indirizzo + risorsa = full\ JID\ (user@unipr.messenger.it/Home)$
L'utilizzo delle risorse è utile per decidere su quale dispositivo collegato al proprio account veicolare la comunicazione.

E' possibile quindi avere più sessioni aperte collegate allo stesso account, ma su dispositivi diversi e ad ogni dispositivo è associata un univoca risorsa.

Sessione XMPP

E' costituita da "stream" di XML su connessioni TCP di tipo long-lived. Differentemente da quello che accade in una normale comunicazione TCP, nella quale una connessione viene instaurata, utilizzata e poi abbattuta, una connessione TCP long-lived, una volta aperta, rimane tale per scambi di stream XML a oltranza fino a quando il client non decide di disconnettersi dal server. Questo tipo di comunicazione rispecchia a pieno il concetto di Real Time Communication, permettendo agli utenti del servizio di essere sempre connessi.
Una sessione XMPP è composta da tre tipi di snippets, chiamati XML Stanzas:

1) <*presence*> : permette di vedere se un utente online (*type="available"* o *"unavailable"*). Questa funzionalità deve essere accettata per ogni utente a cui interessa sapere lo stato di un altro, tramite un *presence subscription* bidirezionale.

<presence from="alice@wonderland.lit" to="sister@realworld.lit" type="subscribe"/>
<presence from="sister@realworld.lit" to="alice@wonderland.lit" type="subscribed"/>

2) *<message>* : type (normal, chat, groupchat, headline, error). Specifica il tipo di messaggio che viene scambiato tra utenti. Contiene le informazioni del sorgente, destinatario e il contenuto.

3) *<iq>* (Info query) : può essere di quattro tipi (get, set, result, error): Viene utilizzata sia dal client che dal server per inviare informazioni di gestione o accedere a servizi.

Sviluppo della piattaforma (SMACK)

Il client UniprMessenger è sviluppato in Java come progetto Maven. L'interfaccia è costruita utilizzando la libreria JavaFX, scaricabile dal marketplace di Eclipse IDE. Il cuore pulsante dell'applicazione è la libreria Smack.

Smack è una libreria open source per sviluppare client XMPP. E' scritta in java ed è progettata per essere integrata in programmi più grandi, con l'obiettivo di fornire al programmatore un supporto completo a tutte le funzionalità di XMPP.

Smack ha il suo centro nella classe ***XMPPConnection***, che si occupa,come dice il nome, di fornire una connessione a server XMPP. Tramite *XMPPConnection* è possibile effettuare la procedura di login, fornendo le credenziali del servizio tramite parametri al costruttore o metodi della classe. *XMPPConnection* fornisce l'accesso alla lista contatti tramite il metodo *XMPPConnection.getRoster()* il quale restituisce un oggetto di tipo Roster.

Roster non solo è una rappresentazione della lista contatti ma anche un vero e proprio collegamento alla stessa: eventuali modifiche al Roster si ripercuotono sul server grazie al lavoro di Smack. Grazie alla classe Roster, le altre entità di Xmpp saranno contenitori "intelligenti", senza che si renda necessaria la scrittura di codice ulteriore per propagare le modifiche al server.

Roster fornisce l'accesso ai singoli contatti sotto forma di oggetti di tipo ***RosterEntry***, anch'essa classe "intelligente" in quanto è possibile, ad esempio, richiedere lo stato aggiornato del contatto mediante una chiamata a metodo in un'istanza.

La classe ***ChatManager***, della quale è possibile ottenere un'istanza mediante il metodo *XMPPConnection.getChatManager()*, è responsabile dell'istanziazione di nuove conversazioni. Una nuova conversazione, o chat, nel gergo Smack, è creabile attraverso il metodo *ChatManager.createChat()*, che restituisce un istanza della classe Chat.
Tale metodo accetta come parametri il JID del contatto da chiamare e un oggetto che implementa l'interfaccia MessageListener. Quest'ultima descrive un cosiddetto event listener (o più concisamente listener), ascoltatore di eventi, che specifica un metodo, processMessage(), che verrà chiamato ogni qualvolta un messaggio verrà ricevuto per quella chat.

La classe Chat consente di gestire una conversazione. In particolare il metodo sicuramente più interessante è quello per spedire messaggi alla controparte: *sendMessage().*

Offline Message Manager
E' una classe che permette di richiedere al server i messaggi indirizzati a un utente mentre era offline.

È possibile associare dei listener anche ad istanze di Roster, tali listener vengono evocati ad esempio in caso di aggiunta di nuovi contatti da parte di connessioni attive in client diversi, oppure al cambiamento di stato dei contatti.
Grazie a questo genere di listener è possibile codificare comportamenti particolari dell'applicazione in reazione a eventi relativi alla lista contatti.

Analisi del traffico di rete (Wireshark)

Utente già registrato (tentativo di Login) - Server (160.78.174.44) Client (160.78.246.44)
Cattura Wireshark in allegato

Connect (8 pacchetti)

- N° 124, 125, 126 → Questi primi tre rappresentano l'instaurazione della connessione TCP, tramite SYN, SYN-ACK, ACK.
- N° 131, 132 → instaurazione Stream XML tra il client e il server. Il client conferma la ricezione con un pacchetto ACK (n° 133).
- N°134 → Il server invia al client la caratteristiche della connessione:
    - algoritmo di compressione (zlib)
    - algoritmo di crittografia (sha-1)
    - Tag Register → se presente, il server permette la registrazione di un nuovo utente dal cliente.
    - Meccanismi di autenticazione accettati dal server (Plain, Scram-Sha-1)

  Il client conferma la ricezione delle caratteristiche con un ACK.

Login

- N° 258 → Il login inizia con un pacchetto XMPP di tipo AUTH (autenticazione) da parte del client contenente le credenziali criptate con algoritmo "SCRAM-SHA-1". Avviene uno scambio di tre pacchetti legate al meccanismo di sicurezza.
- N° 262 → Il server invia il pacchetto contenente la risposta all'autenticazione (SUCCESS credenziali corrette, FAILURE credenziali errate). In caso di credenziali errate, viene chiuso lo stream XML.
- N° 267 → Il client invia IQ Set per richiedere al server la risorsa da associare al proprio indirizzo.
- N° 268 → il server invia al client il risultato della richiesta del pacchetto precedente, facendo il BIND tra il suo indirizzo e la risorsa assegnata.
- UNKNOWN → con il primo pacchetto unknown il client richiede un id per identificare ed eventualmente ripristinare la connessione in caso di errori di rete.

Request for ROSTER

- N°273 → Il Client richiede il “gestore degli amici (ROSTER)” con una query di tipo get.
- N° 274 → Il server risponde con il Roster associato. Successivamente un ACK di conferma dal client

Numero di messaggi sul server

- N°531 → Il client richiede al server quanti sono i messaggi in attesa di invio mentre era offline (iq get disco#info)
- N°532 → Il server risponde con iq type=”result” impostando nel campo *IQ>QUERY>X-DATA>FIELD>VALUE* il numero dei messaggi presenti sul server.

Nel nostra cattura il numero di messaggi da “scaricare” dal server è pari 0, cioè mentre l’utente era offline non ha ricevuto messaggi.
Se ci fossero messaggi in attesa il *VALUE* sarebbe maggiore di 0 e l’applicazione richiederebbe al server di inviare pacchetti di tipo *Message* per ciascun messaggio in attesa.
Per effettuare questa richiesta il client invierebbe un pacchetto *IQ GET OFFLINE.*
Dopo aver ricevuto i messaggi il client invierebbe al server un messaggio di tipo *IQ SET OFFLINE PURGE* per svuotare la memoria dei messaggi in attesa per questo client sul server.

Presence

N°1324 → Il Client invia un messaggio XMPP PRESENCE con valore di *STATUS “Online”,* notificando il suo stato di attività al server.

Logout

Prima di disconnettersi dal server il client invia un messaggio di presenza con *type = unavailable,* notificando il suo stato di offline. Infine viene inviato un pacchetto di Stream END, il quale chiude lo stream XML tra Client e Server e viene effettuato il teardown della connessione TCP con il doppio *two way handshaking* (FIN, ACK, FIN, ACK).