

Chapter 1

Introduction

Cross Domain Communication using XMPP protocol is an application can be used to login into the two popular Social Networking websites: Gmail and Facebook simultaneously or one at a time. It is basically used for achieving inter-domain communication for the purpose of a multi-user chat. A user can chat with his buddies from Gmail as well as Facebook at one place by forming a multi user chat room. This application handles the channeling of messages by distinguishing them at the back end. However, at the front end, every user gets a feel of participating in real-time multi-user chat. An application has also been developed which handles file transfer and multi-user chat for a local XMPP server.

1.1 Purpose

Till date, group chat has been possible only if the members of the group are on one domain. This application has been developed with the intention of extending this functionality of a multi-user chat to users operating on heterogeneous domains. This idea helps in enabling group chats at one place. A message that has to be conveyed in real-time to buddies on different domains can just be sent from one application. This will help in limiting unnecessary usage of bandwidth and waste of time. The file transfer application was developed with the idea of enabling a user to send files at one go to all the members of the multi user chat room. Also, the members of the chat room can simultaneously chat with each other. The user can send a file to a particular user too instead of broadcasting it to everyone in the group.

1.2 Scope

This application allows dual login which enables the user to access contacts from both his Gmail and Facebook accounts simultaneously. The connections to both his accounts are separate and do not interfere with each other. At the same time, they work simultaneously.

The file transfer functionality in the second application enables multi-user file sharing thereby providing the user with the function to send files to multiple people.

1.3 Definitions, Abbreviations

1. XML: eXtensible Markup Language
2. XMPP: eXtensible Messaging and Presence Protocol
3. J2EE: Java 2 Enterprise Edition.
4. SQL: Structured Query Language.
5. API: Application Programming Interface.
6. GUI: Graphical User Interface.
7. SIP: Session Initiation Protocol.
8. JRE: Java run time environment.

1.4 Motivation

The motivation behind this project was to provide a common platform for communication. Instead of logging in to different accounts for communicating with different people, the user should be able to do everything at one place. This platform or chat client allows users to create a group where in members can be using different servers (like Facebook, Gmail or yahoo) and enables group chat between them.

Nowadays we want to be in touch with many people at a time. We don't want to waste our time logging in to different accounts and talking to their respective contacts separately. Sometimes we may want to share files as well as our thoughts with people available to us through different platforms. This chat client helps us achieve the same.

1.5 Problem Definition

This project focuses on achieving cross domain communication for different platforms. Hence, a client logged into one platform (or service provider) will be able to communicate with a client logged into another platform (or service provider). Thus, communication between users irrespective of the domain would become possible. Also, to make this application ubiquitous, the project aims at providing a web interface for the same. The web interface removes the need to have a desktop application installed on the user's system in order to communicate. The project aims at extending text based cross domain communication client to include audio communication as well. This is achieved using the Jingle protocol for file transfer.

Chapter 2

Literature Review

2.1 Introduction

XMPP (Extensible Messaging and Presence Protocol) is an open standard used for achieving real time communication. It uses XML for exchanging messages. XMPP can be viewed as services and applications, from a broader view.

Services provided by XMPP:

1. Channel Encryption
2. Authentication
3. Presence
4. Contact List or Roster
5. One-to-one Messaging
6. Multi-party messaging
7. Notifications

2.2 XMPP Architecture

There are Jabber servers and Jabber clients which implement the XMPP Protocol.

Decentralized architecture is advantageous because there is separation of tasks, simplified functionality of clients, no single point of failure and relatively easier management.

[18]

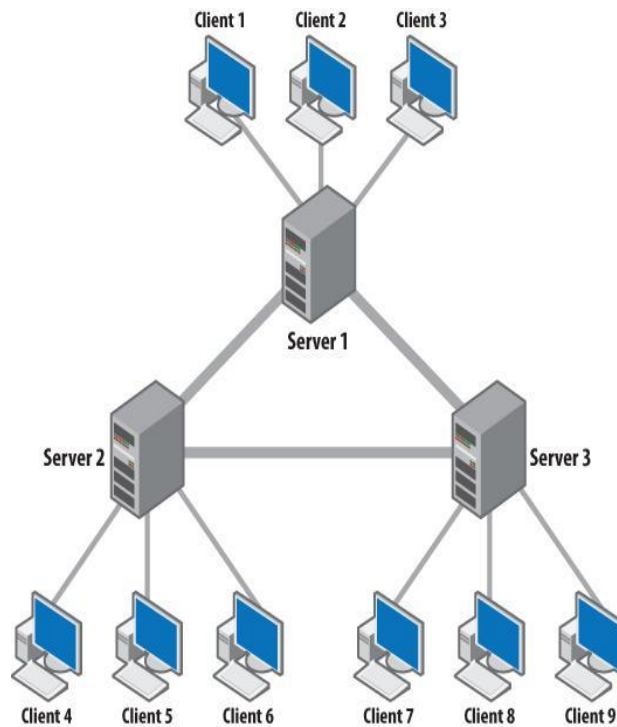


Figure 2.1: XMPP Client-Server Architecture [1]

2.2.1 Terminologies

Addresses: Every XMPP entity has a JabberID which can be either a bare JID (user@domain.tld) or a full JID (user@domain.tld/resource). It uses the Domain Name System. The domain name portion maps to a FQDN. The domain name portion is case-insensitive.

Resources: When a client connects to an XMPP server, the client chooses or the server assigns a resource ID for that particular connection. This is used to route traffic to that connection among many open connections. The resource can be used to refer to a computer or a specific location.

XML Streaming:

To start a session, a TCP connection is established between the XMPP client and XMPP server. After this, a stream is negotiated between the two entities. Once a stream is negotiated, three XML snippets (stanzas) can be exchanged in either direction: <presence/>, <message/> and <iq/>

2.2.2 Stanzas

<message/> stanza:

This is used to send messages from one entity to another. It uses fire-and-forget mechanism. The type attribute distinguishes the message stanzas. They can be one of the following: normal, chat, group chat, headline or error.[10]

Example of a message stanza:

```
<message from="abc@xyz.com" to="pqr@xyz.com" type="chat">
<body> Hi !! </body></message>
```

<presence/> stanza:

This is used to determine which entities are online and available i.e to determine an entity's current network availability. It makes use of the subscription model. The authorization process is known as presence subscription. Presence information is described in roster which is a presence enabled contact list. A presence stanza that does not possess a 'type' attribute is used to signal to the server that the sender is online and available for communication. If included, the 'type' attribute specifies a lack of availability, a request to manage a subscription to another entity's presence, a request for another entity's current presence, or an error related to a previously-sent presence stanza.[10]

Example of a presence stanza:

```
<presence from="abc@xyz.com" to="pqr@xyz.com" type="available" />
```

<iq/> stanza:

Info/Query stanza uses the request-response structure. An iq request must always be followed by an iq response. The requests and responses are tracked using the id attribute. The type attribute takes four different values: get, set, result or error. [10]

Example of an iq stanza:

```
<iq from="abc@xyz.com/wr" id="a12345" to="abc@xyz.com" type="get">
<query xmlns="jabber:iq:roster"/>
</iq>
```

2.2.3 Subscription Handshake

To request someone's presence you send that person a subscription request.

```
<presence from="abc@xyz.com" to="pqr@xyz.com" type="subscribe" />
```

When recipient receives this request, he can either approve it or deny it.

```
<presence from="pqr@xyz.com" to="abc@xyz.com" type="subscribed" />
```

```
<presence from="pqr@xyz.com" to="abc@xyz.com" type="unsubscribe" />
```

The subscription handshake is bidirectional. Therefore the recipient also sends a subscription request to the sender. When you send a <presence/> stanza to your server, the server sends a presence stanza to each user subscribed to your presence. This is how everyone knows you're online. To know which users are online, your server sends a presence probe. The users which are authorized receive presence notifications.[12]

```
<presence from="abc@xyz.com" to="pqr@xyz.com" /> ,
```

```
<presence from="abc@xyz.com" to="stu@xyz.com" /> , etc.
```

```
<presence from="abc@xyz.com" to="pqr@xyz.com" type="probe" />
```

[20]

2.2.4 Chat states

- Starting: The conversation is started, but you haven't joined in yet.
- Active: You are actively involved in the conversation.
- Composing: You are actively composing a message.
- Paused: You started composing a message, but then stopped.
- Inactive: You haven't contributed to the conversation for some period of time.
- Gone: Your involvement with the conversation has effectively ended [1]

2.3 Smack API

Smack is a library for communicating with XMPP servers to perform real-time communications, including instant messaging and group chat.

Class ConnectionConfiguration

public abstract class ConnectionConfiguration extends Object

Configuration to use while establishing the connection to the server.

//Creates a ConnectionConfiguration object specifying the desired configuration of hostname, port number and service to be used.

ConnectionConfiguration config=new ConnectionConfiguration(hostname, port, service);

Methods of ConnectionConfiguration class:

public CharSequence getUsername()

Returns the username to use when trying to reconnect to the server.

public String getPassword()

Returns the password to use when trying to reconnect to the server.

public String getResource()

Returns the resource to use when trying to reconnect to the server.

Interface XMPPConnection

public interface XMPPConnection

The XMPPConnection interface provides an interface for connections to an XMPP server and implements shared methods which are used by the different types of connections. To create a connection to an XMPP server a simple usage of this API might look like the following:

```
// Create a connection to the XMPP server.  
XMPPConnection con = new XMPPConnection(config);  
// Connect to the server  
con.connect();  
// Most servers require you to login before performing other tasks.  
con.login("jsmith", "mypass");  
// Start a new conversation with John Doe and send him a message.
```

Methods of XMPPConnection interface:

String getUser()

Returns the full XMPP address of the user that is logged in to the connection or null if not logged in yet. An XMPP address is in the form username@server/resource.

String getHost()

Returns the host name of the server where the XMPP server is running. This would be the IP address of the server or a name that may be resolved by a DNS server.

void sendPacket(Stanza packet) throws SmackException.NotConnectedException

Sends the specified stanza(/packet) to the server.

Parameters: packet - the stanza(/packet) to send.

Throws: SmackException.NotConnectedException

void addPacketListener(StanzaListener packetListener, StanzaFilter packetFilter)

Parameters: packetListener - the stanza(/packet) listener to notify of new received packets,
packetFilter - the stanza(/packet) filter to use.

Class ChatManager

public class **ChatManager** extends Object

The chat manager keeps track of references to all current chats. It will not hold any references in memory on its own so it is necessary to keep a reference to the chat object itself. To be made aware of new chats, register a listener by calling addChatListener(ChatManagerListener).

```
Chat chat= ChatManager.getInstanceFor(con).createChat("jdoe@igniterealtime.org", new  
MessageListener() {
```

```
    public void processMessage(Chat chat, Message message) {  
        // Print out any messages we get back to standard out.  
        System.out.println("Received message: " + message);  
    }  
});  
chat.sendMessage("Howdy!");  
// Disconnect from the server  
con.disconnect();
```

Methods of class ChatManager:

addChatListener

```
public void addChatListener(ChatManagerListener listener)
```

Register a new listener with the ChatManager to receive events related to chats.

Parameters:listener - the listener.

createChat

```
public ChatcreateChat(String userJID, MessageListener listener)
```

Creates a new chat and returns it.

Parameters:userJID - the user this chat is with, listener - the listener which will listen for new messages from this chat.

Returns: the created chat.

Class Roster

`public class Roster extends Manager`

Represents a user's roster, which is the collection of users a person receives presence updates for. Roster items are categorized into groups for easier management.

Others users may attempt to subscribe to this user using a subscription request. Three modes are supported for handling these requests:

- `accept_all` -- accept all subscription requests.
- `reject_all` -- reject all subscription requests.
- `manual` -- manually process all subscription requests.

Methods of class Roster:

`public boolean addRosterListener(RosterListener rosterListener)`

Adds a listener to this roster. The listener will be fired anytime one or more changes to the roster are pushed from the server.

Parameters: `rosterListener` - a roster listener.

Returns: `true` if the listener was not already added.

`public void createEntry(String user, String name, String[] groups) throws`

`SmackException.NotLoggedInException, SmackException.NoResponseException,`

`XMPPException.XMPPErrorException, SmackException.NotConnectedException`

Creates a new roster entry and presence subscription. The server will asynchronously update the roster with the subscription status.

Parameters: `user` - the user. (e.g. `john@jabber.org`), `name` - the nickname of the user, `groups` - the list of group names the entry will belong to, or null if the the roster entry won't belong to a group.

Throws: `SmackException.NoResponseException` - if there was no response from the server,
`XMPPException.XMPPErrorException` - if an XMPP exception occurs,
`SmackException.NotLoggedInException` - If not logged in,
`SmackException.NotConnectedException`

```
public RosterGroup createGroup(String name)
```

Creates a new group.

Parameters: name - the name of the group.

Returns: a new group, or null if the group already exists

Throws: IllegalStateException - if logged in anonymously

```
public RosterEntry getEntry(String user)
```

Returns the roster entry associated with the given XMPP address or null if the user is not an entry in the roster.

Parameters: user - the XMPP address of the user (eg "jsmith@example.com"). The address could be in any valid format (e.g. "domain/resource", "user@domain" or "user@domain/resource").

Returns: the roster entry or null if it does not exist.

```
public Presence getPresence(String user)
```

Returns the presence info for a particular user. If the user is offline, or if no presence data is available (such as when you are not subscribed to the user's presence updates), unavailable presence will be returned.

If the user has several presences (one for each resource), then the presence with highest priority will be returned. If multiple presences have the same priority, the one with the "most available" presence mode will be returned. In order, that's free to chat, available, away, extended away, and do not disturb.

Parameters: user - an XMPP ID. The address could be in any valid format (e.g. "domain/resource", "user@domain" or "user@domain/resource"). Any resource information that's part of the ID will be discarded.

Returns: the user's current presence, or unavailable presence if the user is offline or if no presence information is available.

Interface MessageListener

```
void processMessage(Message message)
```

```
{ //
}
```

Class Presence

```
public final class Presence
```

```
extends Stanza
```

```
implements TypedCloneable<Presence>
```

Represents XMPP presence packets. Every presence stanza(/packet) has a type, which is one of the following values:

- available -- (Default) indicates the user is available to receive messages.
- unavailable -- the user is unavailable to receive messages.
- subscribe -- request subscription to recipient's presence.
- subscribed -- grant subscription to sender's presence.
- unsubscribe -- request removal of subscription to sender's presence.
- unsubscribed -- grant removal of subscription to sender's presence.
- error -- the presence stanza(/packet) contains an error message.

Methods of class Presence:

```
public Presence.Type getType()
```

Returns the type of this presence packet.

Returns:

The type of the presence packet.

```
public void setStatus(String status)
```

Sets the status message of the presence update. The status is free-form text describing a user's presence (i.e., "gone to lunch").

Parameters:

status - the status message.[14]

2.4 Openfire Server

Openfire (previously known as **Wildfire**, and **Jive Messenger**) is an instant messaging (IM) and groupchat server that uses XMPPserver written in Java and licensed under the Apache License 2.0.

Features

Openfire supports the following features:

- Web-based administration panel
- Plugin interface
- Customizable
- SSL/TLS support^[3]
- User-friendly web interface and guided installation
- LDAP connectivity
- Platform independent, pure Java
- Full integration with Spark (XMPP client)
- Can support more than 50,000 concurrent users
- Database connectivity for storing messages and user details.[16]

2.5 Audio Communication

2.5.1 Session Initiation Protocol

The Session Initiation Protocol (SIP) is a communications protocol for signaling and controlling multimedia communication sessions. The most common applications of SIP are in Internet telephony for voice and video calls, as well as instant messaging all over Internet Protocol (IP) networks.

The protocol defines the messages that are sent between endpoints, which govern establishment, termination and other essential elements of a call. SIP is an application layer protocol designed to be independent of the underlying transport layer.

Media identification and negotiation is achieved with the Session Description Protocol (SDP). For secure transmissions of SIP messages, the protocol may be encrypted with Transport Layer Security (TLS).[2]

2.5.2 Protocol Operation

SIP can be used for two-party (unicast) or multiparty (multicast) sessions. SIP employs design elements similar to the HTTP request/response transaction model. Each transaction consists of a client request that invokes a particular method or function on the server and at least one response.

Each resource of a SIP network, such as a user agent or a voicemail box, is identified by a uniform resource identifier (URI). The URI scheme used for SIP is sip: and a typical SIP URI is of the form: [sip:username:password@host:port](#).

If secure transmission is required, the scheme sips: is used and mandates that each hop over which the request is forwarded up to the target domain must be secured with Transport Layer Security (TLS). SIP clients typically use TCP or UDP. Port 5060 is commonly used for non-encrypted signaling traffic whereas port 5061 is typically used for traffic encrypted with Transport Layer Security (TLS).[11]

SIP is primarily used in setting up and tearing down voice or video calls. It also allows changing addresses or ports, inviting more participants, and adding or deleting media streams.

The voice and video stream communications in SIP applications are carried over the Real-time Transport Protocol (RTP). Parameters (port numbers, protocols, codecs) for these media streams are defined and negotiated using the Session Description Protocol (SDP), which is transported in the SIP packet body. SIP is only involved in the signaling portion of a communication session. The features that permit familiar telephone-like operations: dialing a number, causing a phone to ring, hearing ringback tones or a busy signal - are performed by proxy servers and user agents.

SIP is a client-server protocol, however most SIP-enabled devices may perform both the client and the server role. In general, session initiator is a client, and the call recipient performs the server function. SIP features are implemented in the communicating endpoints.[3]

2.5.3 Network Elements

- A **SIP User Agent** can perform the role of a *User Agent Client* (UAC), which sends SIP requests, and the *User Agent Server* (UAS), which receives the requests and returns a SIP response. These roles of UAC and UAS only last for the duration of a SIP transaction. The user agent may identify itself using a message header field 'User-Agent', containing a text description of the software/hardware/product involved.
- **The proxy server** is an intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, meaning that its job is to ensure that a request is sent to another entity closer to the targeted user.
- A **registrars** is a SIP endpoint that accepts REGISTER requests and places the information it receives in those requests into a location service for the domain it handles. The location service links one or more IP addresses to the SIP URI of the registering agent.
- **Redirect server:** A user agent server that generates 3xx (Redirection) responses to requests it receives, directing the client to contact an alternate set of URIs. The redirect server allows proxy servers to direct SIP session invitations to external domains.
- **Session border controllers** serve as middle boxes between UA and SIP servers for various types of functions, including network topology hiding, and assistance in NAT traversal.
- **Gateways** can be used to interface a SIP network to other networks, such as the public switched telephone network, which use different protocols or technologies.[2]

Illustration:

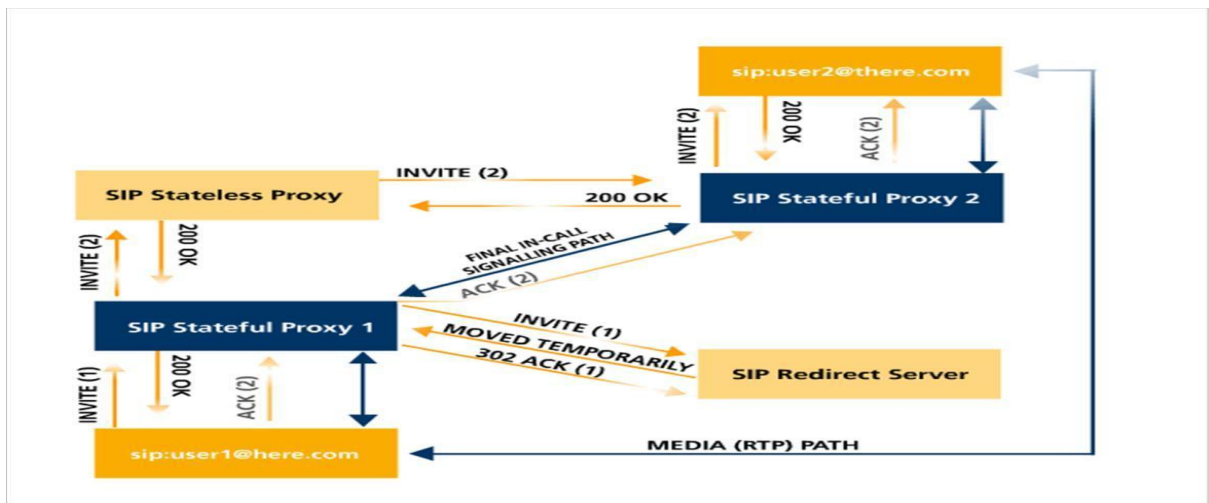


Figure 2.2: SIP Illustration [2]

2.5.4 AsteriskNOW Server

Asterisk is an open source framework for building communications applications. Asterisk turns an ordinary computer into a communications server. Asterisk powers IP PBX systems, VoIP gateways, conference servers and other custom solutions. It is used by small businesses, large businesses, call centres, carriers and government agencies, worldwide. Asterisk is free and open source.[4]

AsteriskNOW is an open source software appliance, a customized Linux distribution that includes Asterisk, the Asterisk GUI, and all other software needed for an Asterisk system.

AsteriskNOW is the premier, ready-to-run distribution of open source Asterisk. AsteriskNOW is an ISO image that installs Linux, Asterisk and the FreePBX GUI in a single, simple install. The Asterisk GUI gives you the ability to easily configure your Asterisk system without being a technical expert.[5]

AsteriskNOW makes it easy to create custom telephony solutions by automatically installing the "plumbing." AsteriskNOW was built for application developers, systems integrators, students, hackers and others who want to create custom solutions with Asterisk. It is freely available for use at home, at school or at work. With AsteriskNOW, application developers and integrators can concentrate on building solutions, not maintaining the plumbing.[4]

2.5.5 Jingle

In essence, Jingle provides a way for Jabber clients to set up, manage, and tear down multimedia sessions. Such sessions can support a wide range of application types (such as voice chat, video chat, and file transfer) and use a wide range of media transport methods (such as TCP, UDP, RTP, or even in-band XMPP itself). The signaling to establish a Jingle session is sent over XMPP, and typically the media is sent directly peer-to-peer or through a media relay. Jingle provides a pluggable framework for both application types and media transports; in the case of voice and video chat, a Jingle negotiation usually results in use of the Real-time Transport Protocol (RTP) as the media transport and thus is compatible with existing multimedia technologies such as the Session Initiation Protocol (SIP). Furthermore, the semantics of Jingle signaling was designed to be consistent with both SIP and the Session Description Protocol (SDP), thus making it straightforward to provide signaling gateways between XMPP networks and SIP networks.[6]

Jingle (XEP-0166) can be used to initiate and negotiate a wide range of peer-to-peer sessions. One session type of interest is media such as voice or video. This document specifies an application format for negotiating Jingle media sessions, where the media is exchanged over the Real-time Transport Protocol

The Jingle application format defined herein is designed to meet the following requirements:

1. Enable negotiation of parameters necessary for media sessions using the Real-time Transport Protocol (RTP).
2. Map these parameters to Session Description Protocol to enable interoperability.
3. Define informational messages related to typical RTP uses such as audio chat and video chat (e.g., ringing, on hold, on mute).[7]

```
<descriptionxmlns='urn:xmpp:jingle:apps:rtp:1'media='audio'>
  <payload-typeid='96'name='speex'clockrate='16000'/>
  <payload-typeid='97'name='speex'clockrate='8000'/>
  <payload-typeid='18'name='G729'/>
  <payload-typeid='103'name='L16'clockrate='16000'channels='2'/>
  <payload-typeid='98'name='x-ISAC'clockrate='8000'/>
  <payload-typeid='102'name='iLBC'/>
  <payload-typeid='4'name='G723'/>
  <payload-typeid='0'name='PCMU'clockrate='16000'/>
  <payload-typeid='8'name='PCMA'/>
  <payload-typeid='13'name='CN'/>
</description>
```

Figure 2.3: XML Format for Jingle [7]

2.6 Web Interface

2.6.1 BOSH Server

When a user interacts with other users and applications on the network (through a computer), there are two basic types of interaction:

Client Initiated: Here the user makes a decision to do something. The client (the software on the computer that the user is using) will send protocol to a server on the network. Typically this forms a request/response pattern, with the client making a request and the server providing a response, although XMPP allows for other patterns.

Server Initiated: Here the user is passive. Something happens remotely; A server sends something to a client; The user is alerted or informed in some way. This may form either a request/response pattern or a pure server-push where no response is expected.

Why a Different Approach is needed for XMPP?

Server initiated interactions are central to the way XMPP operates. In particular:

- Messages may arrive, either from one of your buddies or in a MUC room.
- The online status of one of your buddies may change.

Unless you have an appropriate browser window open, there is no mechanism for the user to be alerted in a standard Web browser. This is a reason why many XMPP users (and IM users in general) prefer to use a desktop client. A desktop application can give useful alerting, even when the primary window is minimized or obscured (e.g., by a "pop up" alert).

A more general problem is that the core Web (HTTP) protocol is client (browser) initiated. This means that there is no basic mechanism for a server web application to display information on the user's screen in response to a server initiated event. This means that the simple application server architecture used for client initiated protocol will not work for XMPP. [17]

BOSH (Bidirectional streams Over Synchronous HTTP) specified in XEP-0124 is the standardized way to do XMPP over HTTP. BOSH is simply a means of carrying the XMPP protocol over HTTP. BOSH defines a simple framing to use these packets in HTTP protocol. For client initiated protocol the client simply sends packets over HTTP (using HTTP post). Server initiated protocol is handled by a technique called 'long polling'. In long polling, the web browser initiates a standard request, but does not expect a response back immediately. When the server has data to send, it will send it as a response to the request, and the client will immediately issue another request, thus keeping a request pending for the server to respond to.

General purpose chat applications are generally desktop. BOSH is typically used for more specialized requirements, such as:

- Specialized MUC clients, such as speeqe, which provides an easy Web interface to chat rooms.
- Multi-user games. XMPP is ideal infrastructure for large scale gaming. ChessPark was an interesting example of a game system built on XMPP.
- Jappix is an open social network based on XMPP using a BOSH interface.[8]

2.6.2 Strophe.js

Strophe.js is an XMPP library for JavaScript. Its primary purpose is to enable web-based, real-time XMPP applications that run in any browser.

The Strophe Family

There are currently two members of the Strophe family of libraries.

- **Strophe.js**

Strophe.js is a JavaScript implementation targeting browser-based clients. It uses BOSH, a binding of XMPP to HTTP using long polling and WebSockets, a full-duplex single socket connection to a server. Strophe.js makes creating real-time web applications easy.

- **libstrophe**

libstrophe is a C library for XMPP clients and components. It has very minimal dependencies and was designed with both POSIX and Windows systems in mind.

Strophe.js is an XMPP library for JavaScript. Its primary purpose is to enable web-based, real-time XMPP applications that run in any browser. Since JavaScript had no facilities for persistent TCP connections, this library uses Bidirectional-streams Over Synchronous HTTP (BOSH) to emulate a persistent, stateful, two-way connection to an XMPP server.

Strophe.js API

1. **\$build:** function \$build(name, attrs)

Create a Strophe. Builder. This is an alias for 'new Strophe.Builder(name, attrs)'.

2. **\$msg:** function \$msg(attrs)

Create a Strophe. Builder with a <message/> element as the root.

3. **\$iq:** function \$iq(attrs)

Create a Strophe. Builder with an <iq/> element as the root.

4. **\$pres:** function \$pres(attrs)

Create a Strophe. Builder with a <presence/> element as the root. [9]

Chapter 3

System Analysis and Design

3.1 Desktop Application for Text-based Cross Domain Communication

Our chat client is developed using Java. The library used was the Smack API which is the XMPP library for Java. The chat client handles all the functionality which happens at the back end by performing dual login. One user logs in through Gmail and also simultaneously logs in through Facebook. Two XMPPConnection objects are made for handling two login simultaneously attempts. The two XMPP connections work separately without interfering with each other so that the messages are routed correctly. Each XMPPConnection has its own Roster (buddy list) which gets updated as presence changes.

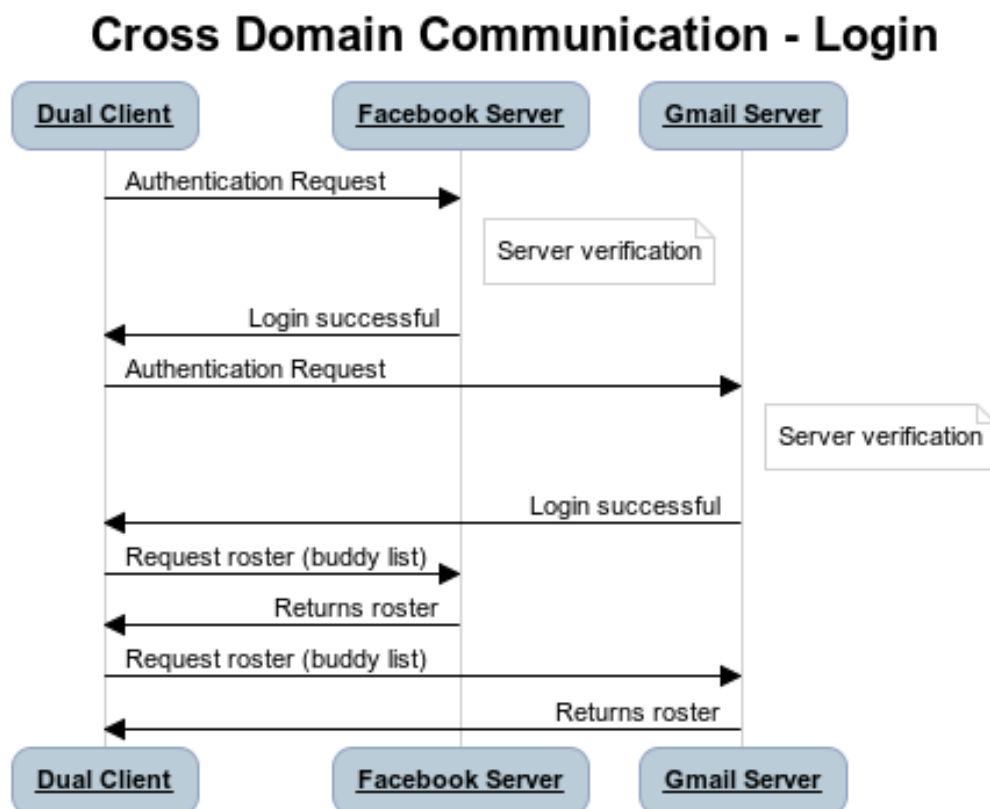


Figure 3.1 Sequence diagram for authentication

A group is dynamically created each time a user wants to perform a multi user chat. The group created includes his contacts from Gmail and (or) also from Facebook. The user initiating the multi user chat can include contacts from any of his rosters. The members in the group may or may not be each other's friends. Still they are able to communicate with each other and a virtual chat room is created.

Cross Domain Communication - Group Creation

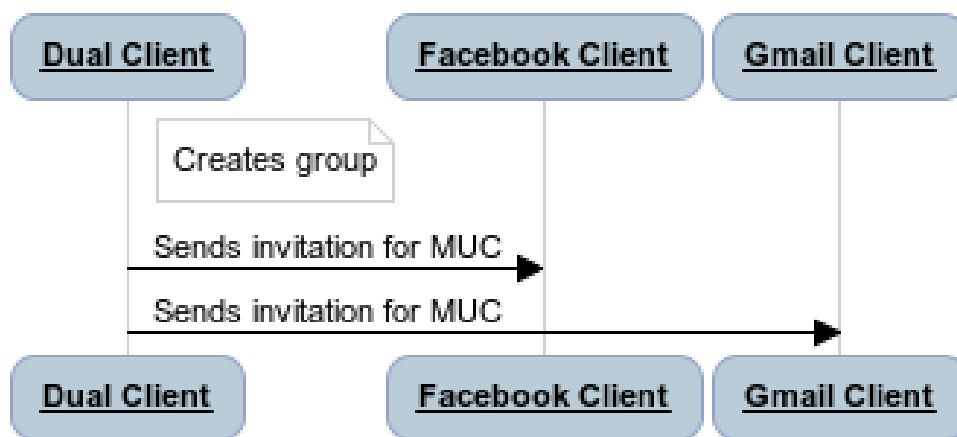


Figure 3.2 Sequence diagram for group creation

The user who has created the chat broadcasts his message to all the members of the chat room. At the back end, the messages are sent to Facebook and Gmail users separately using the separate XMPPConnection objects mentioned earlier. When any particular member replies, the message is received by the sender which again broadcasts it to all the members. This gives a virtual appearance of a chat room where one member sends a message which is received by all others. [21]

Cross Domain Communication - Message Transfer

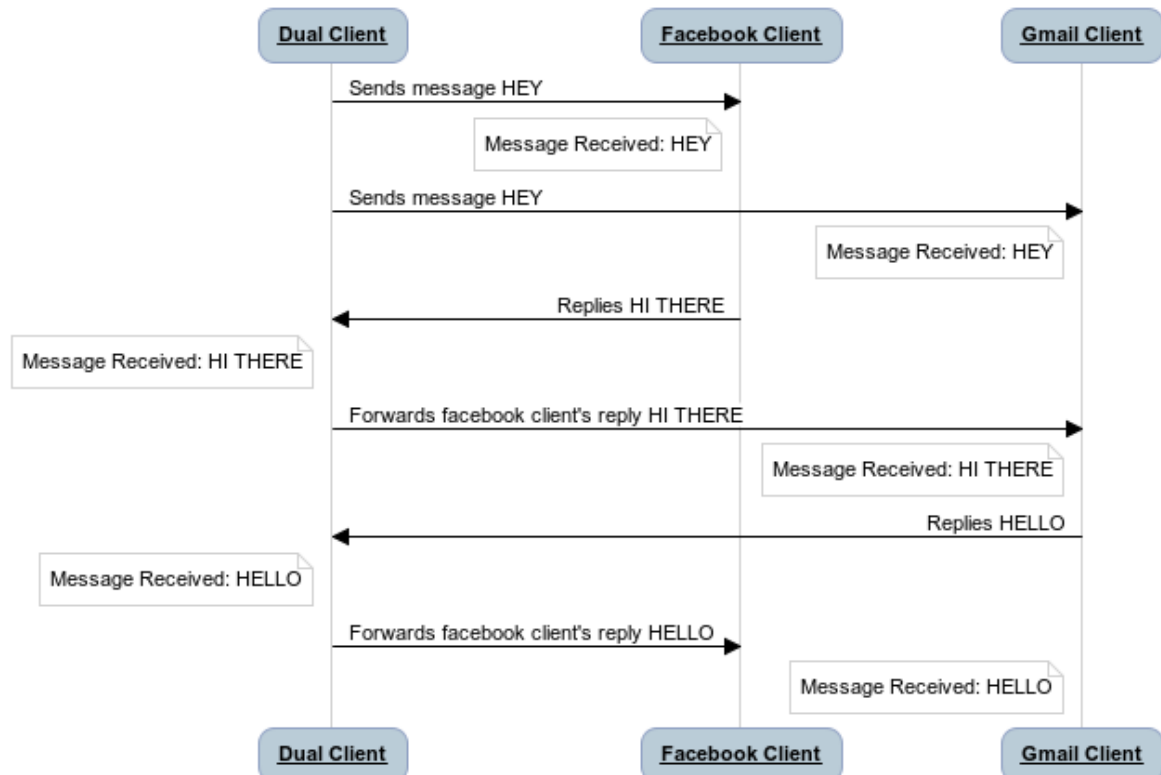


Figure 3.3 Sequence diagram for message transfer

The group members other than the chat room owner have to login through only one account either Facebook or Gmail. They can communicate with other members logged in through either Facebook or Gmail. They just need to be in the buddy list of the chat room owner and not necessarily other members. The member uses XMPPConnection interface for connections to an XMPP server and logs into his/her account. He receives the messages sent by other members of the chat room using the same connection. While replying in the chat room he/she sends message directly to the owner and the owner broadcasts the message on this member's behalf. The owner differentiates this message from the single user chat messages and broadcasts it to the chat room occupants.

3.2 Limitations and Problems

3.2.1 Limitation of Desktop

By definition, a desktop application means any software that can be installed on a single computer (laptop or a desktop) and used to perform specific tasks.

Some of the major disadvantages of desktop application are as follows:

- Maintenance - Desktop applications are to be installed separately on each computer. Also updating the applications is cumbersome with desktop applications as it needs to be done on every single computer
- Ease of use - Desktop applications are confined to a physical location and hence have usability constraint.
- Platform dependent- Desktop applications have traditionally been limited by the hardware on which they are run. They must be developed for and installed on a particular operating system, and may have strict hardware requirements that must be met to ensure that they function correctly. Updates to the applications must be applied by the user directly to their installation, and may require hardware upgrades or other changes in order to work. This hardware dependence, as well as the legacy of mainframe terminal applications, has typically limited the level of complexity in user interfaces for desktop applications
- Challenging to deploy to large number of users.
- Piracy is higher in case of desktop applications

3.2.2 Web based application over Desktop based

Web based applications are ubiquitous in nature and hence can be accessed anywhere with the help of Internet unlike desktop based

Browser based software never requires installation processes or hard drive space. It lives in a virtual cloud in the Internet and this means that whenever you launch it, it always has the latest version

It is platform independent unlike desktop applications and therefore a lot more flexible

Given that the browser is now the platform, operational support costs and maintenance for Web Application providers drop substantially.

It is a lot more easy to increase the reach of a web based application as there isn't a need to install it in every single device (as in the case of desktop based application) and will also take lesser amount of time.

3.2.3 Web Interface and its problems

When a user interacts with other users and applications on the network (through a computer), there are two basic types of interaction:

- **Client Initiated:** Here the user makes a decision to do something. The client (the software on the computer that the user is using) will send protocol to a server on the network. Typically this forms a request/response pattern, with the client making a request and the server providing a response, although XMPP allows for other patterns.
- **Server Initiated:** Here the user is passive. Something happens remotely; A server sends something to a client; The user is alerted or informed in some way. This may form either a request/response pattern or a pure server-push where no response is expected.

Why a Different Approach is needed for XMPP?

Server initiated interactions are central to the way XMPP operates. In particular:

- Messages may arrive, either from one of your buddies.
- The online status of one of your buddies may change.

Unless you have an appropriate browser window open, there is no mechanism for the user to be alerted in a standard Web browser. This is a reason why many XMPP users (and IM users in general) prefer to use a desktop client. A desktop application can give useful alerting, even when the primary window is minimized or obscured (e.g., by a "pop up" alert).

A more general problem is that the core Web (HTTP) protocol is client (browser) initiated. This means that there is no basic mechanism for a server web application to display information on the user's screen in response to a server initiated event. This means that the simple application server architecture used for client initiated protocol will not work for XMPP.[18]

3.3 Web Application for Text-based Cross Domain Communication

3.3.1 Strophe.js

Strophe.js is an XMPP library for JavaScript. Its primary purpose is to enable web-based, real-time XMPP applications that run in any browser.

While most XMPP libraries and implementations are focused on chat-based applications, Strophe takes a grander view. It has been used to implement real-time games, notification systems, search engines, as well as traditional instant messaging.

The Strophe Family

There are currently two members of the Strophe family of libraries.

- **Strophe.js**

Strophe.js is a JavaScript implementation targeting browser-based clients. It uses BOSH, a binding of XMPP to HTTP using long polling and WebSockets, a full-duplex single socket connection to a server. Strophe.js makes creating real-time web applications easy.

- **libstrophe**

libstrophe is a C library for XMPP clients and components. It has very minimal dependencies and was designed with both POSIX and Windows systems in mind. Strophe.js is an XMPP library for JavaScript. Its primary purpose is to enable web-based, real-time XMPP applications that run in any browser. Since JavaScript has no facilities for persistent TCP connections, this library uses Bidirectional-streams Over Synchronous HTTP (BOSH) to emulate a persistent, stateful, two-way connection to an XMPP server.

3.3.2 Problems with Strophe.js

A. Connection with Openfire Server:

1. Installing, Configuring and Testing Openfire Server on local machine:

Openfire is a real time collaboration (RTC) server licensed under the Open Source Apache License. It uses the only widely adopted open protocol for instant messaging, XMPP (also called Jabber). Openfire is incredibly easy to setup and administer, but offers rock-solid security and performance. Openfire is an IM server that uses XMPP protocol. As a prerequisite, OpenJDK is a must have in the machine.

Download `openfire_3_7_1.tar.gz` package. Extract the archive:

```
tar -xzf openfire_3_0_0.tar.gz. Move extracted folder to /opt: mv openfire /opt
```

The command to start openfire server is:

```
$ sudo /opt/openfire/bin/openfire start.
```

Access openfire through: <http://localhost:9090>

It will take you through Openfire configuration wizard. Create a MySQL database named 'openfire'. Under Database Settings, in Database URL enter: `jdbc:mysql://localhost:3306/openfire`.

In order to test openfire, you must create a user id. To do this, log in to the administrative console, click Users and Create New User. Once the user is created, you can test your Openfire server using an XMPP Client (Pidgin, Gajim, etc) and configuring the URL and port of your server.

2. Configuring BOSH:

BOSH (Bidirectional streams Over Synchronous HTTP) specified in XEP-0124 is the standardized way to do XMPP over HTTP. BOSH is simply a means of carrying the XMPP protocol over HTTP. The XML XMPP packets used in BOSH are the same as the ones used

when standard XMPP is run over TCP. BOSH defines a simple framing to use these packets in HTTP protocol. For client initiated protocol the client simply sends packets over HTTP (using HTTP post). Server initiated protocol is handled by a technique called 'long polling'. In long polling, the web browser initiates a standard request, but does not expect a response back immediately. When the server has data to send, it will send it as a response to the request, and the client will immediately issue another request, thus keeping a request pending for the server to respond to.

3. Connecting with Strophe.js:

[Strophe](#) is an XMPP library for Javascript which allows you to connect from a Web browser to an XMPP server. Download the latest [version](#) of this library and unzip it in your workspace. This package contains a couple of examples that can be used as reference. In my case, I will use `examples/basic.html`.

B. Connection with Facebook

1. Configuring Punjab Server as BOSH Connection Manager:

Punjab is sort of a black box. It has a xmlrpc api that client's make method calls to and it then converts them into jabber messages and returns them based on the methods the client's call. What goes on in between does not really matter to the client developer.

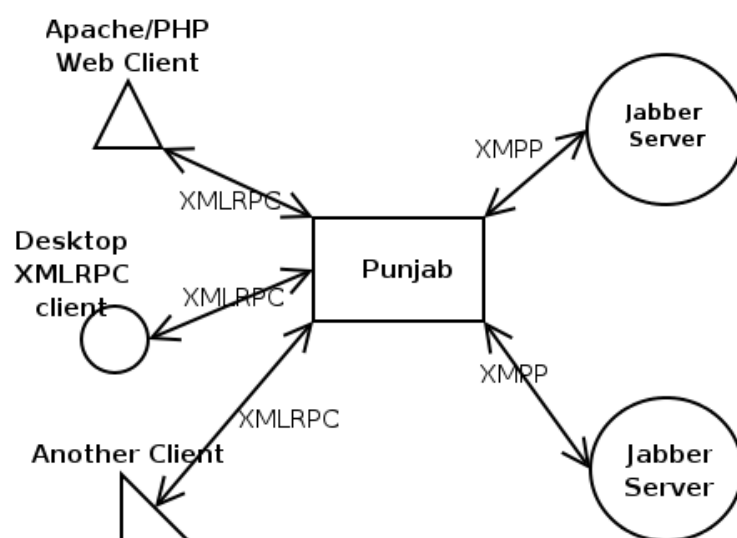


Figure 3.4 Punjab server

A standalone BOSH Server i.e. Punjab was installed for connection to Facebook's XMPP servers using strophe.js. Punjab was created to allow stateless systems a way to interact with [jabber](#), a very stateful system. Jabber or XMPP is a powerful, extendable, and easy to use xml messaging protocol. It is mostly used for instant messaging, but can be used for many other applications.

2. Connecting with Strophe.js:

The connection could not be established even after repeated trials to connect Facebook XMPP Server using strophe.js and the configured Punjab BOSH Connection Manager. The reason behind it is the deprecation of the Facebook Chat API which does not allow to connect to the Facebook XMPP Servers.

C. Connection with Gmail

1. Configuring Punjab Server as BOSH Connection Manager:

Same procedure as in 'Connection with Facebook'

2. Connecting with Strophe.js:

Since Google has stopped GTalk and switched over to Hangouts, the XMPP connection to Google Server is denied using Strophe.js library as XMPP no longer is the underlying protocol.

3.4 Cross Domain Communication using Openfire Server

3.4.1 Cross Domain Communication for text using Web Interface

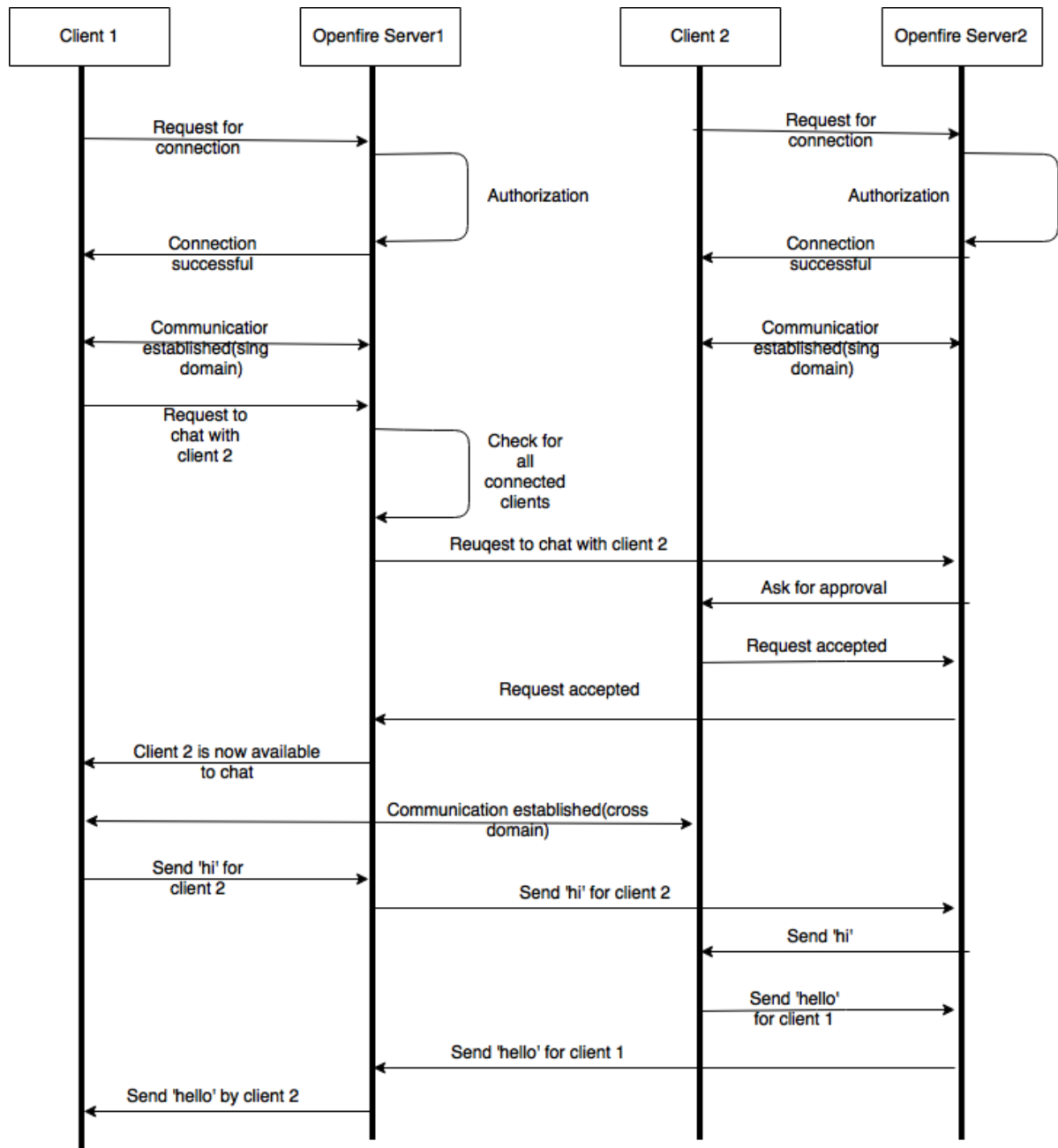


Figure 3.5 Sequence diagram for text based cross domain communication over Web interface

3.4.2 Cross Domain Communication for Audio using Desktop

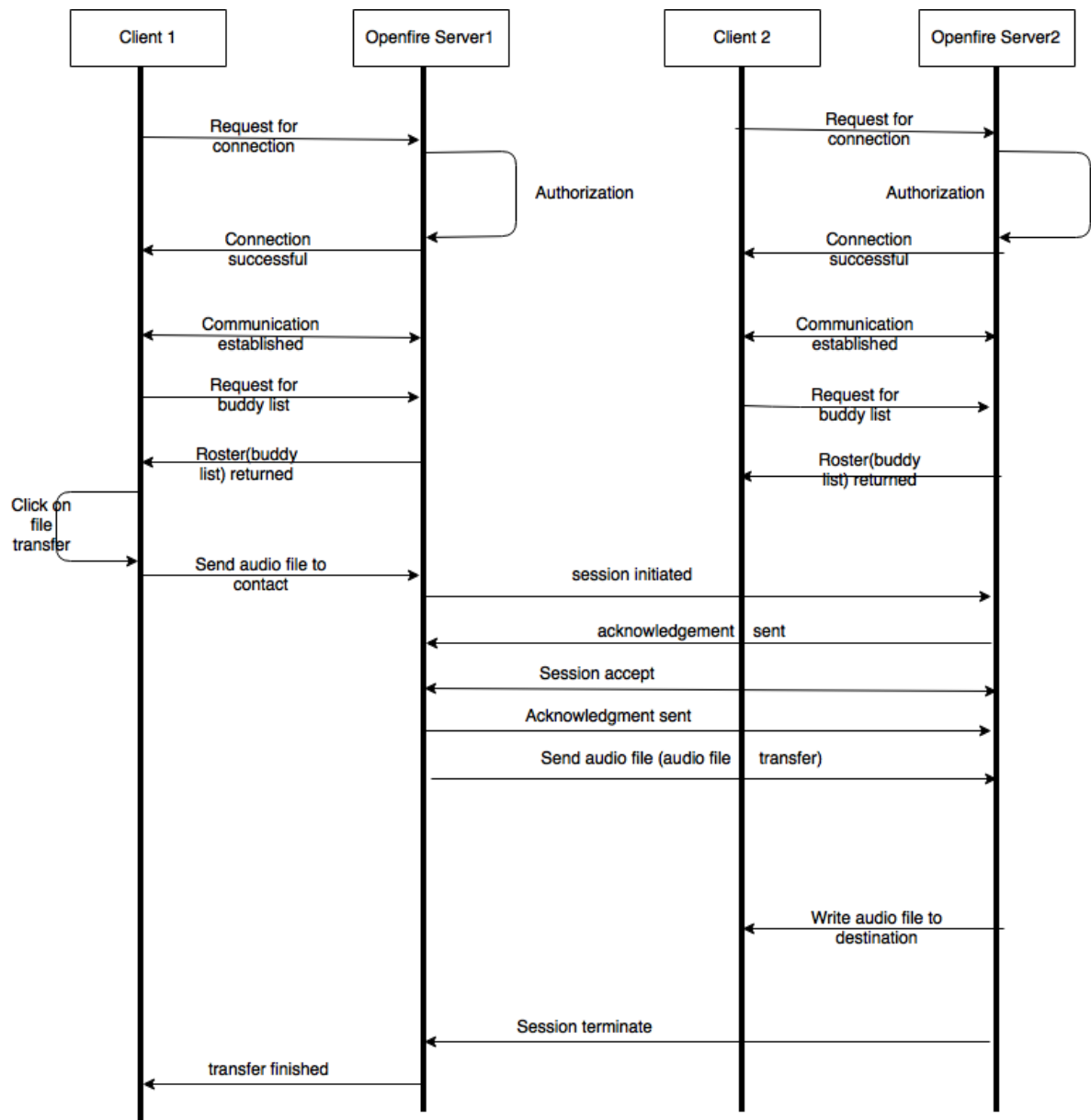


Figure 3.6 Sequence diagram for desktop based audio communication

3.5 XMPP Integration with Asterisk

XMPP stands for eXtensible Messaging and Presence Protocol. It is a widely used communication protocol. Asterisk is open source telephony switching exchange service for Linux. We have used Openfire – an open source XMPP Server and FreePBX install on CentOS 6.5.

Step 1: Creating user in xmpp server

Login to openfire server. You will find the welcome screen as in image below and then click on user/session tab.

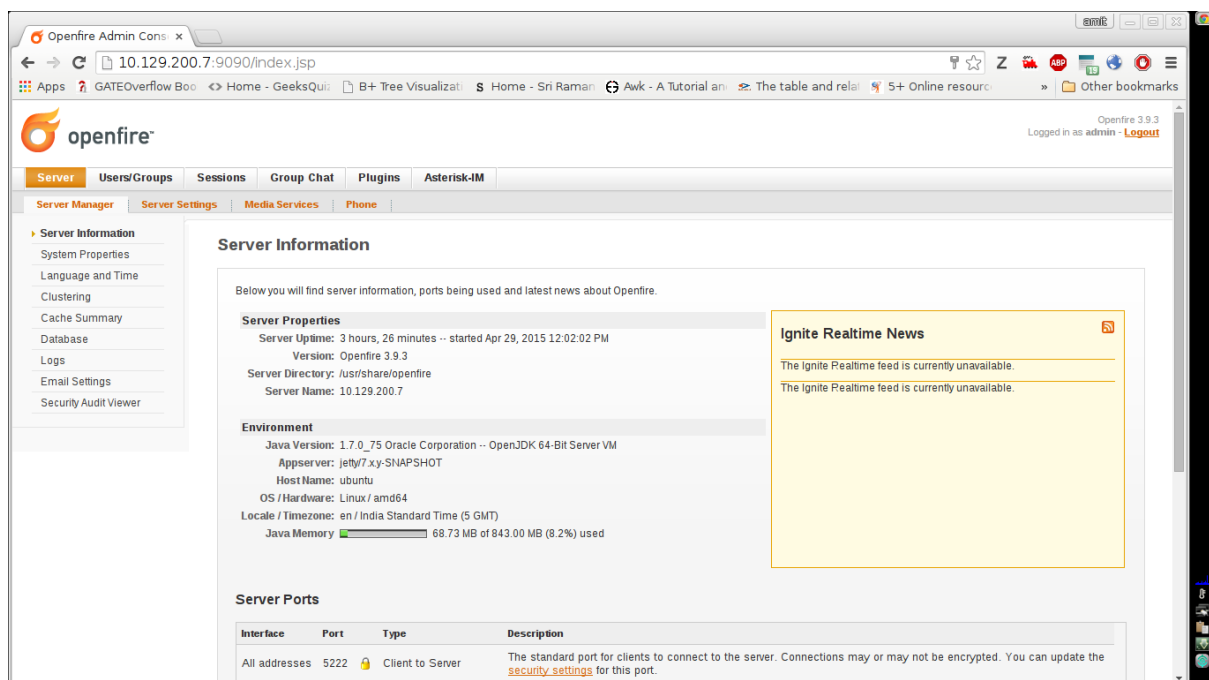


Figure 3.7 Creating user on Openfire server

Now provide the user information and click on create user as in image below.

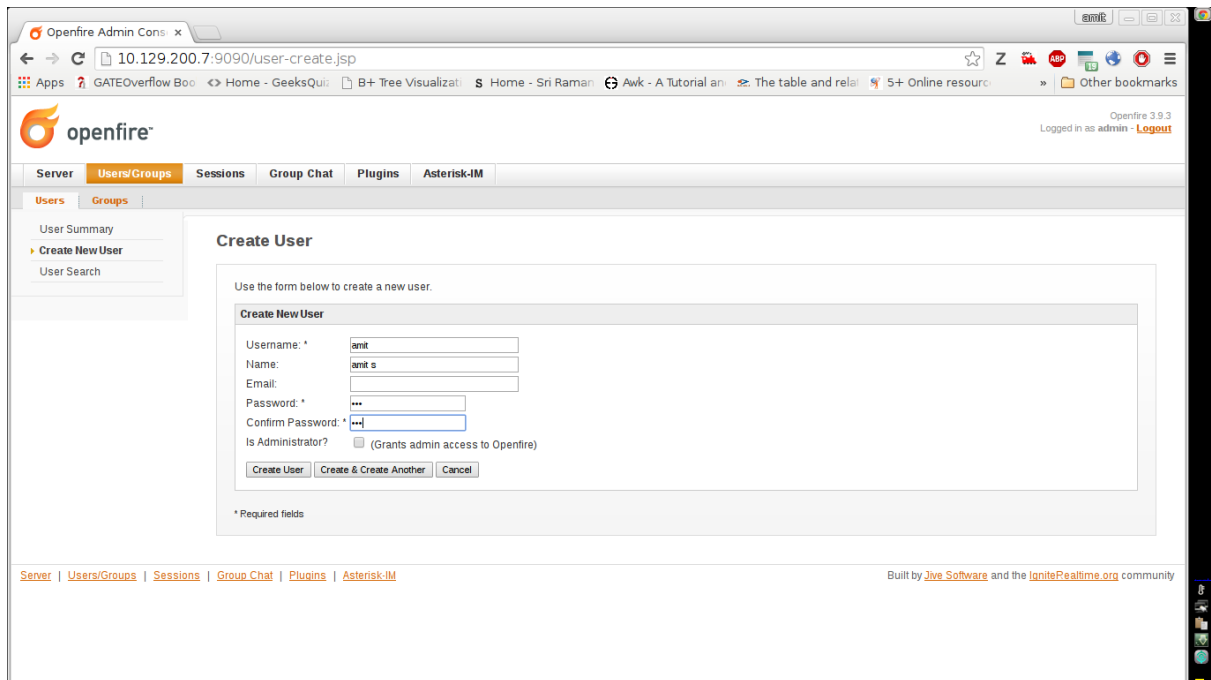


Figure 3.8 Create user page

Step 2: Starting Spark xmpp client

You will find the spark client starts as in image below.

Provide user id, password which you created in Openfire server and give server ip.

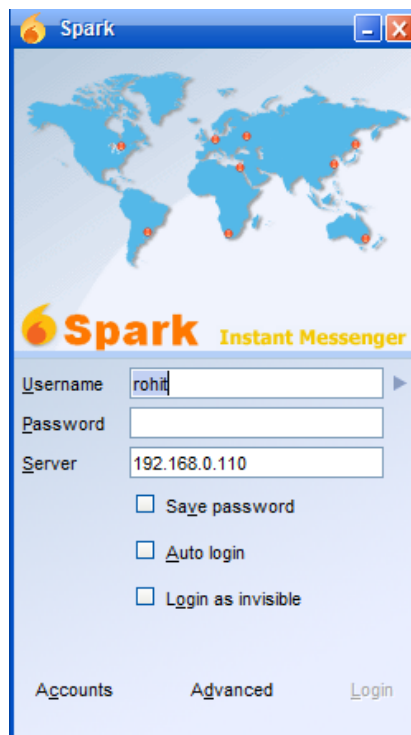


Figure 3.9 Spark client UI

Step 3: Enable SIP plugin to openfire

Step 4: Configure SIP user on asterisk

Now we need to configure sip.conf and extension.conf file to switch the call between asterisk and xmpp client.

sip.conf configuration

```
[100]
username=100
type=friend
secret=123
record_out=never
record_in=never
qualify=no
port=5060
call-limit=4
canreinivite=no
disallow=all
allow=all
```

extension.conf configuration

```
exten => 100,1,Dial(SIP/100,20)
```

The above settings allow sip client to established call to xmpp client which is spark.

Step 5: Configure SIP phone on openfire

On your openfire interface go to Phone tab and then click on add new phone mapping and provide the details.

SIP Phone Settings

XMPP username => user name of your openfire user.

SIP username => user name on your asterisk server which we configure in sip.conf

Authorization username => same as SIP username

PASSWORD => password of your SIP user.

SERVER => IP of asterisk server

Step 6: *Make a call from SIP phone*

Now from the SIP phone the extension number 100 is dialled which is of the XMPP client. Once the call is accepted it will get connected.

Thus, we have successfully integrated XMPP (Openfire) with SIP (Asterisk).

Chapter 4

Implementation and Testing

4.1 Implementation: Web Application

The web application is used as front-end for clients connected to OpenFire Server. It allows users to connect to OpenFire, add buddies by sending them request and also communicate with them. Clients connected to single domain can communicate with clients connected on another domain by sending them requests on the server which they are connected to. The web application uses HTML, JavaScript and CSS as key technologies. It uses strophe.js library for connection between clients and OpenFire Server. The different methods for presence notification, presence change notification, retrieving roster list and for sending and receiving messages are handled by the strophe.js library. The different modules for all this functions are explained below using the component diagram:

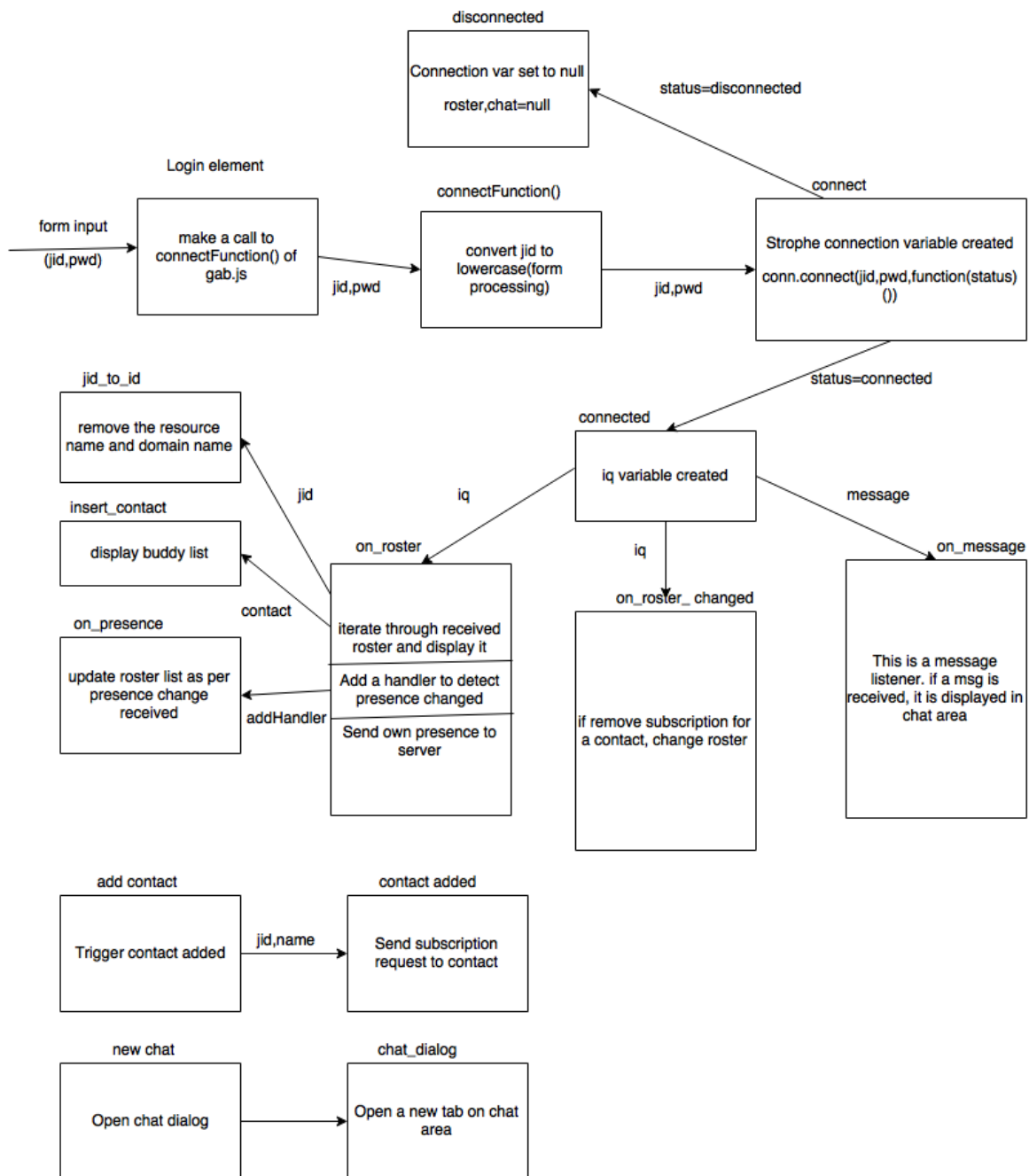


Figure 4.1: Component diagram for Web Application

4.2 Functional Testing: Web Application

1. Test Scenario:

The web application which is used to connect users to XMPP Server i.e. OpenFire Server and communicate with other clients connected cross-domain should have a facility for authorization for users using their JID and Password.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	Authorization and security of users connecting to OpenFire Server	Application shows a 'Login' Page	Enter a legitimate user's JID with proper Domain name and Password	User logs in successfully and becomes online. The web application shows the user's buddy list
ii.	Authorization and security of users connecting to OpenFire Server	Application shows a 'Login' Page	Enter a JID with improper Domain name and/or Password	User buddy list is empty and is prompted to enter correct JID and Password

Screenshots:

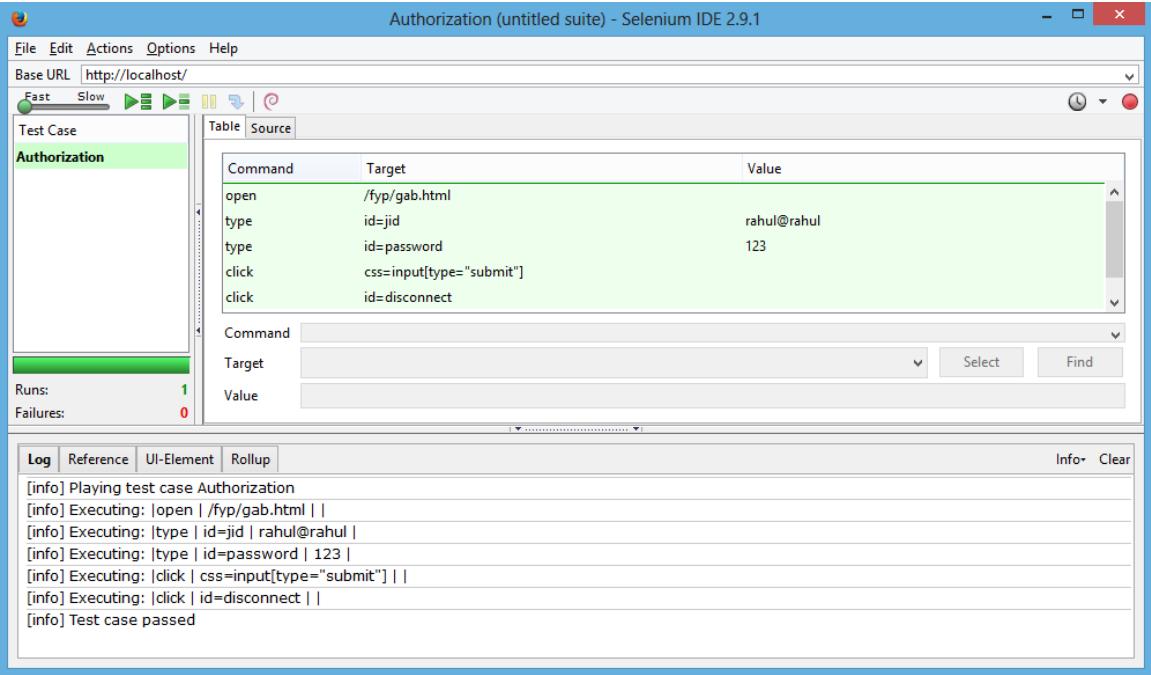


Figure 4.2: Authorization Test Case

Cross Domain Communication

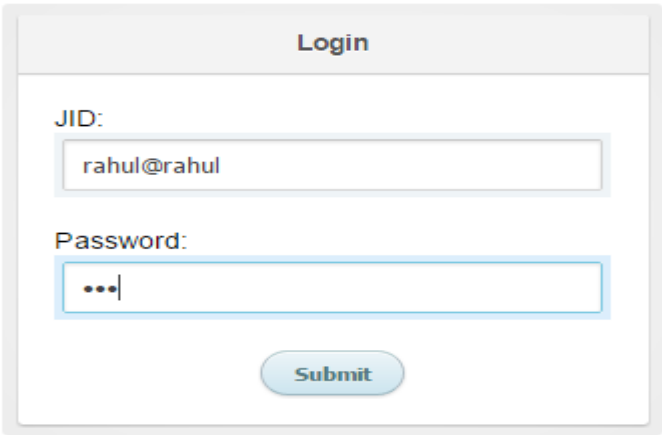


Figure 4.3: Login Page

2. Test Scenario :

The users who can successfully log into the system should be able to see their buddy list and can chat with them. Users should be indicated about the online/offline status of their buddies. Also, users can chat with many buddies simultaneously without corruption of the messages.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	Users able to see buddy list with proper online/offline indicator.	Application shows a 'Buddy list'	No input from user side	Application shows all the buddies which user has accepted earlier. In the buddy list, users who are online are marked green and those not, marked red
ii.	Users can start a chat with any of the buddies in the buddy list	Application shows a 'Buddy list'	User clicks a buddy from the buddy list to start communication	Application shows a chat area where users can communicate via messaging with other user displaying the buddy name (and not domain as it is single domain)
iii.	Users can start a chat with any number of buddies in the buddy list simultaneously	Application shows a 'Buddy list'	User clicks two or more buddies from the buddy list to start communication with them simultaneously	Application shows a chat area displaying names (and not domain as it is single domain) of all buddies in different tabs

Screenshots:

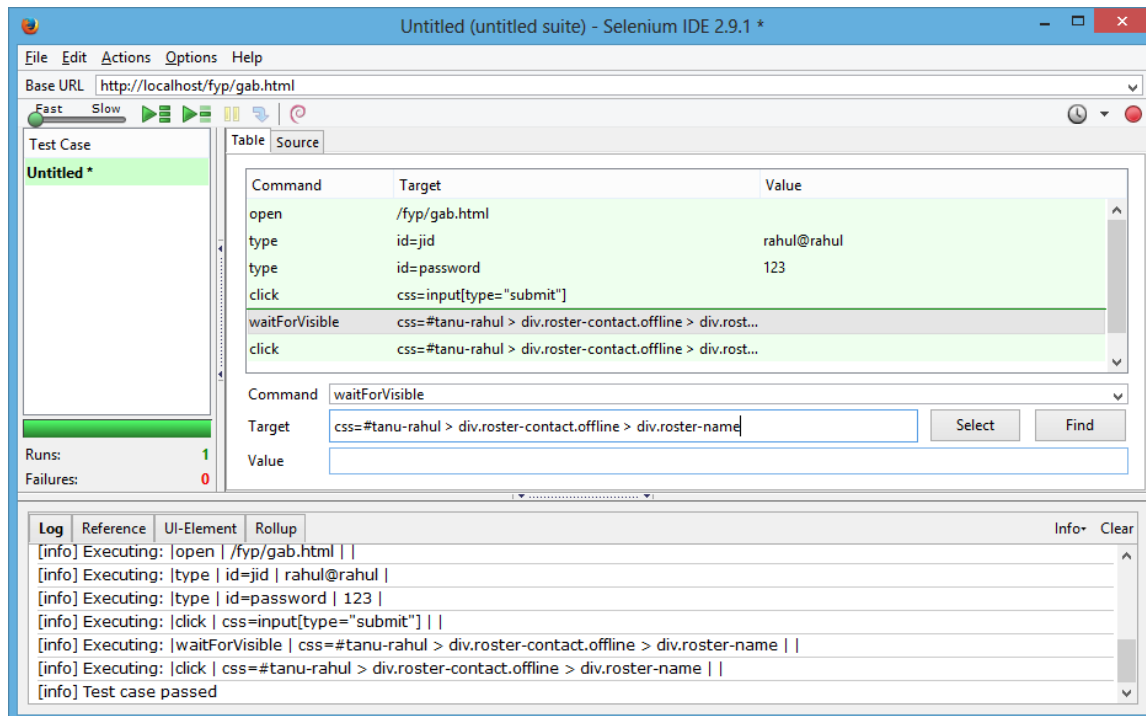


Figure 4.4: Buddy List Test Case

Cross Domain Communication

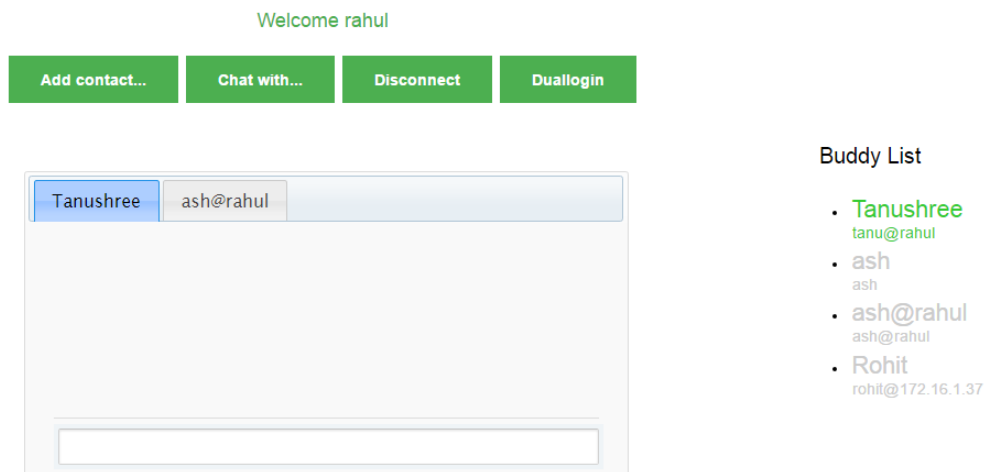


Figure4.5: Buddy List

3. Test Scenario:

User can communicate with other user(s) via the chat area for respective users. User can distinguish between messages by the name prefixed before it and can also view if other user is typing a message.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	User can successfully send and receive messages to and from his buddies (on same domain)	Application shows a chat area with the buddy name	User starts typing in the input box to create a message for his buddy and sends it by pressing the 'Enter' key	Application shows the message received in the message box by prefixing it with the name of the user
ii.	User can view his buddy's activities	Application shows a chat area with the buddy name	No user input	Application shows that the buddy which wants to communicate with you is typing

Screenshots:

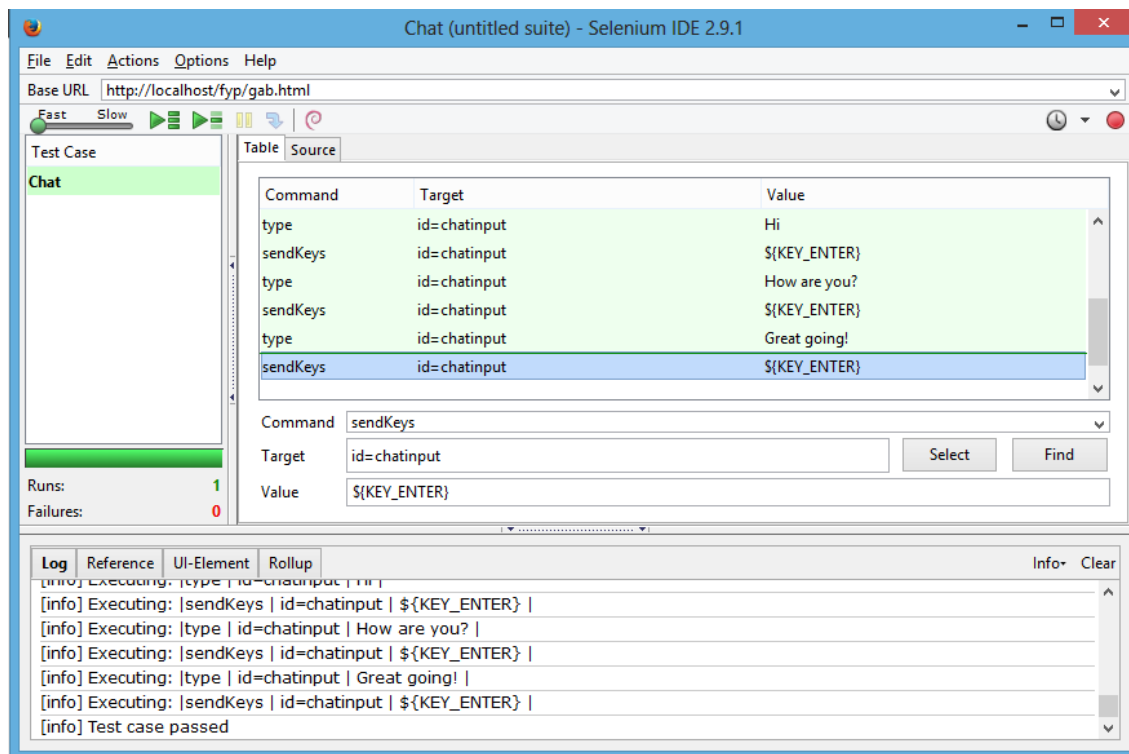
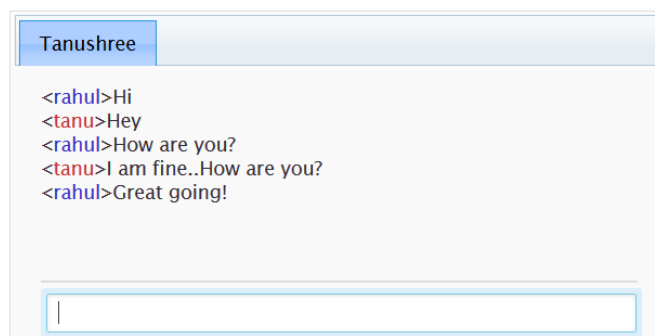


Figure4.6: Chat Test Case

Cross Domain Communication

Welcome rahul



Buddy List

- **Tanushree**
tanu@rahul
- ash
ash
- ash@rahul
ash@rahul
- Rohit
rohit@172.16.1.37

Figure 4.7:Chat using Single Domain

4. Test Scenario:

Users can add buddies which are connected to other server (other domain). They can add them by sending a request to them. Users can accept request from other users and can add them as buddies. Thus, a user can have buddies from same domain as well as from other domain.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	User can add buddies from other domains as well	Application shows the main home page with 'Add a Contact' button	User enters the JID of the other user which he wants to add as his buddy and also types a Name for him	Application sends a request to other user and shows his Name in the buddy list of user who has sent him request
ii.	User can accept requests from users of other domain	Application shows a dialog box containing information who and from where the request has come	User approves or denies the request of the other user by clicking on the appropriate button	In both the cases the user is notified and if approved, they can communicate with each other as other buddies can

Screenshots:

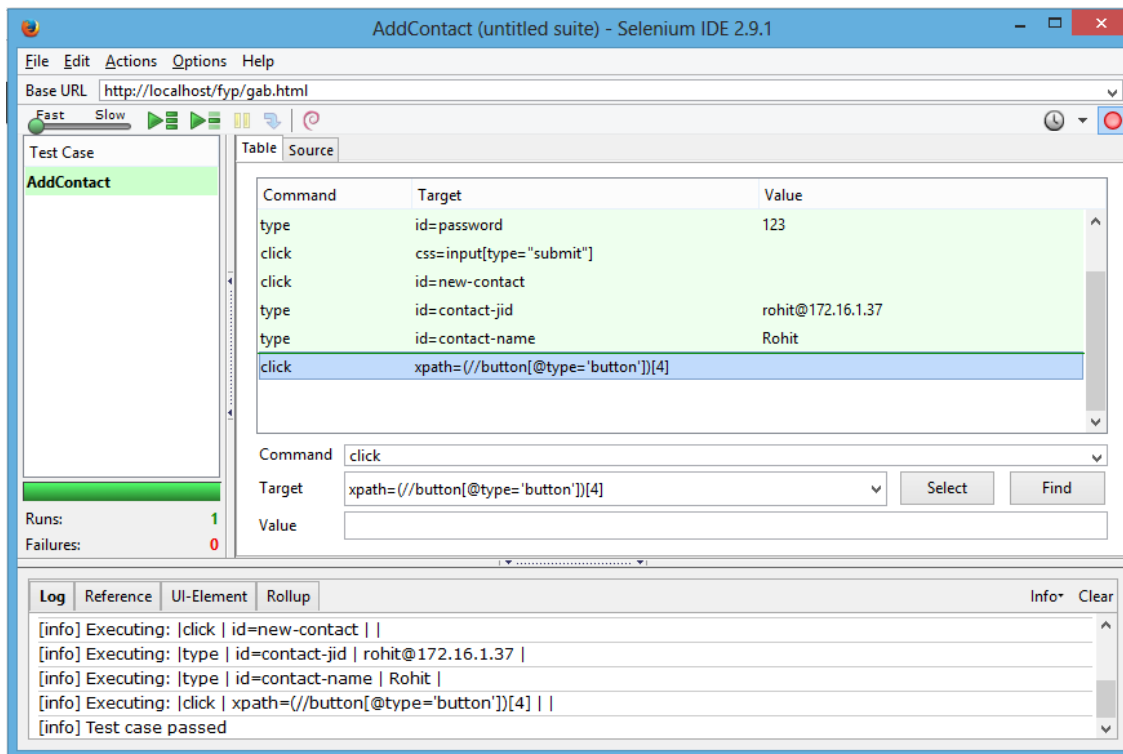


Figure 4.8: Add Contact Test Case

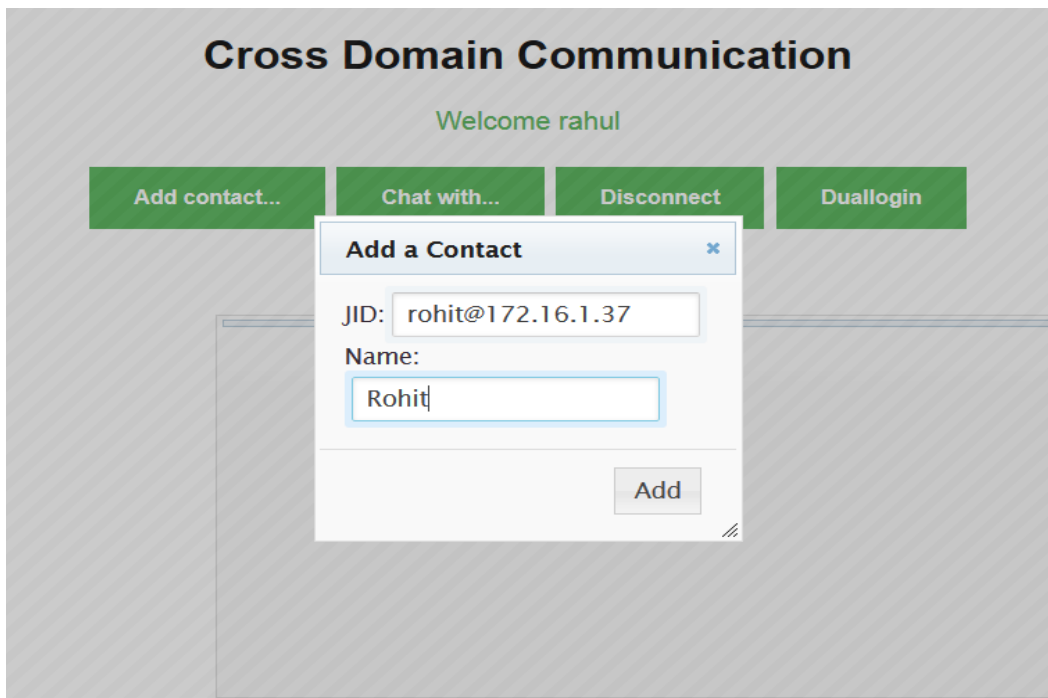


Figure 4.9: Add Contact from another domain

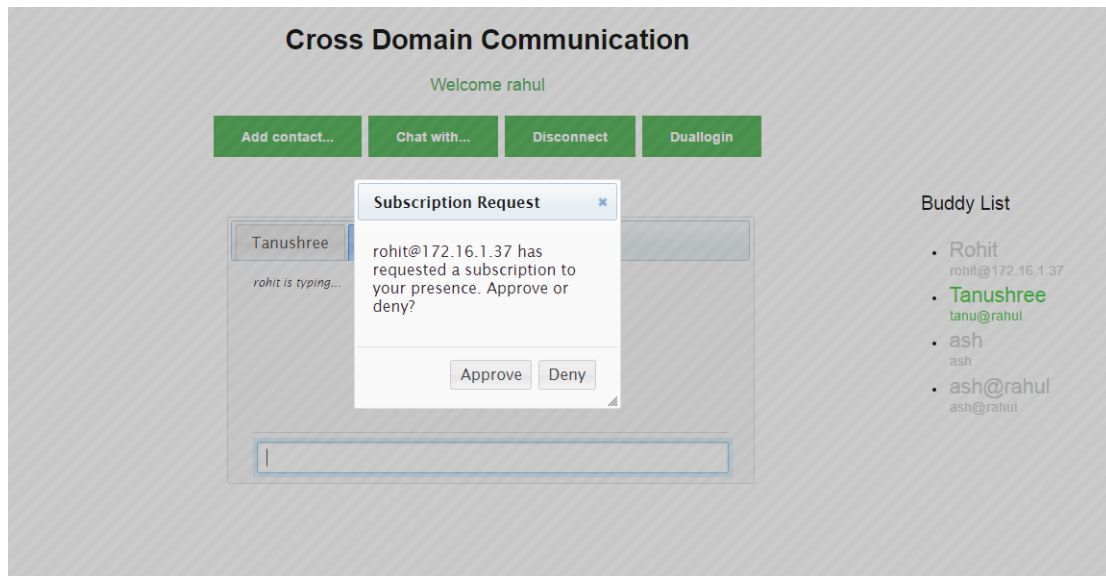


Figure 4.10: Subscription Request

5. Test Scenario:

Users can communicate with buddies from different domain in the similar way as they used to by sending or receiving messages to/from them.

Test Case:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	User can communicate with buddies of different domain	Application shows the chat area of the buddies	User clicks the buddy name from buddy list and start entering message in the input box	Application shows the message received in the message box by prefixing it with the name of the user of other domain

Screenshots:

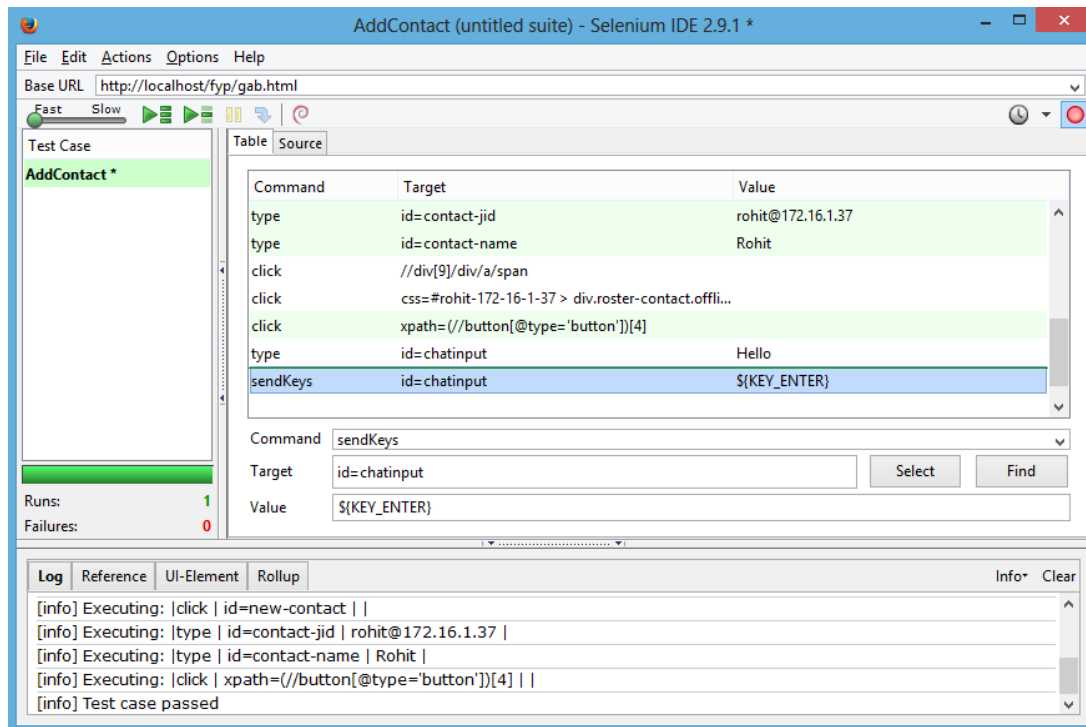


Figure4.11: Chat with other domain user Test Case

Cross Domain Communication

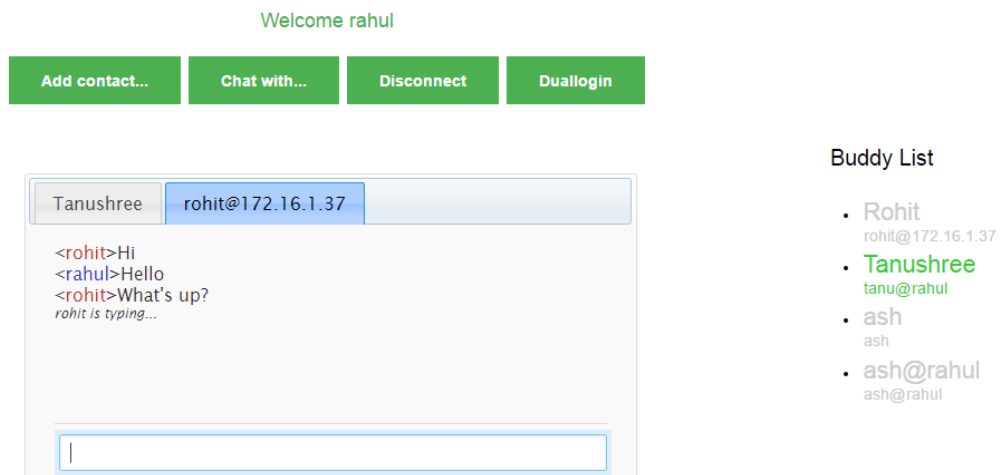


Figure 4.12: Chat with user from other domain

4.3 Implementation: Jingle Application

The desktop application for audio communication using XMPP protocol is used by clients who wish to perform audio communication with each other. These clients are connected to the Openfire server and stored audio files are transmitted using Jingle which is an extension to XMPP for the purpose of file transfer. For communication, two clients connected to the Openfire server have to be in each other's rosters which is achieved by a subscription mechanism. The desktop application uses Java, Smack API and Jingle extension to XMPP protocol as the key technologies. Jingle handles the session establishment and termination needed for file transfer. The different modules for the working of this application are explained below using the component diagram:

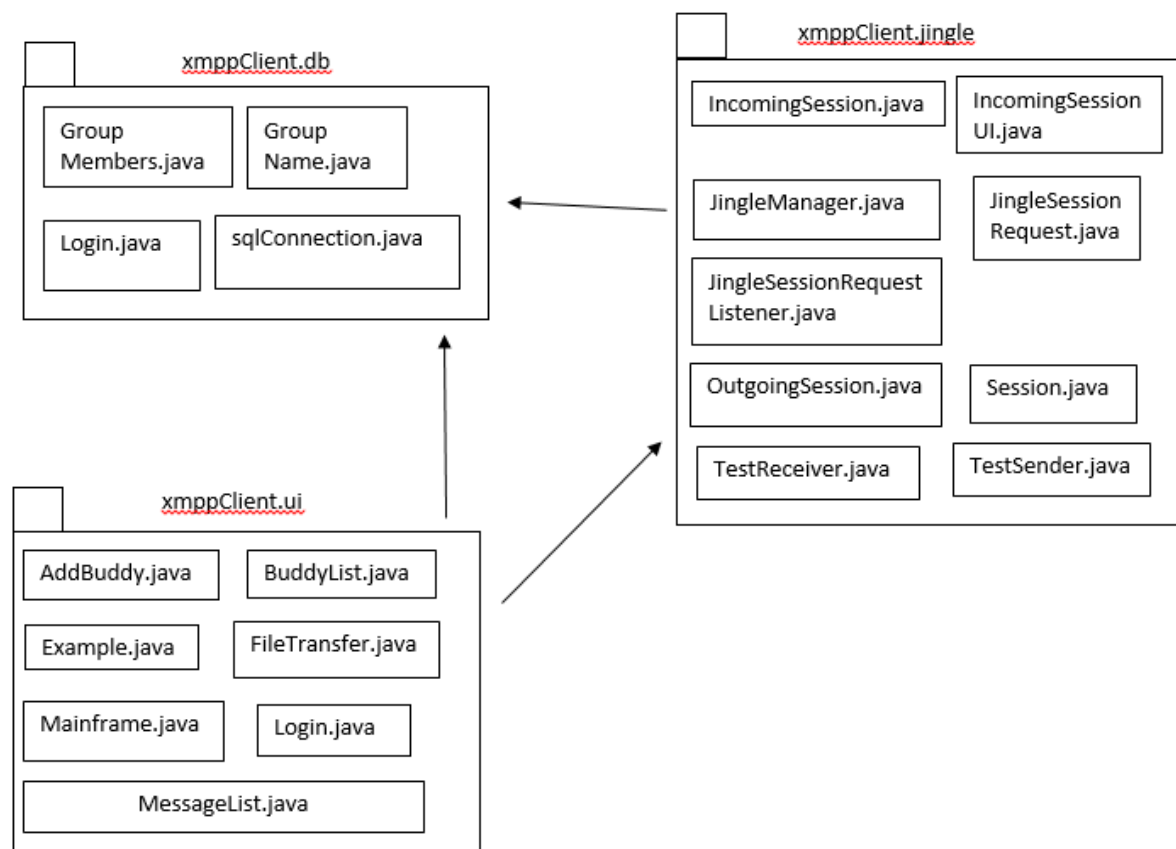


Figure 4.13: Component Diagram for Jingle Application

4.4 Functional Testing: Jingle Application

1. Test Scenario:

The java application which is used to connect users to XMPP Server i.e. OpenFire Server and communicate with other clients connected cross-domain via file transfer should have a facility for authorization for users using their JID and Password.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	Authorization and security of users connecting to OpenFire Server using the java Application	Application shows a 'Login' Page	Enter a legitimate user's JID with proper Domain name and Password	User logs in successfully and becomes online. The java application shows the user's buddy list
ii.	Authorization and security of users connecting to OpenFire Server using the Java Application	Application shows a 'Login' Page	Enter a JID with improper Domain name and/or Password	The Application does not move to next page and the user is prompted to enter correct JID and Password

Screenshots:

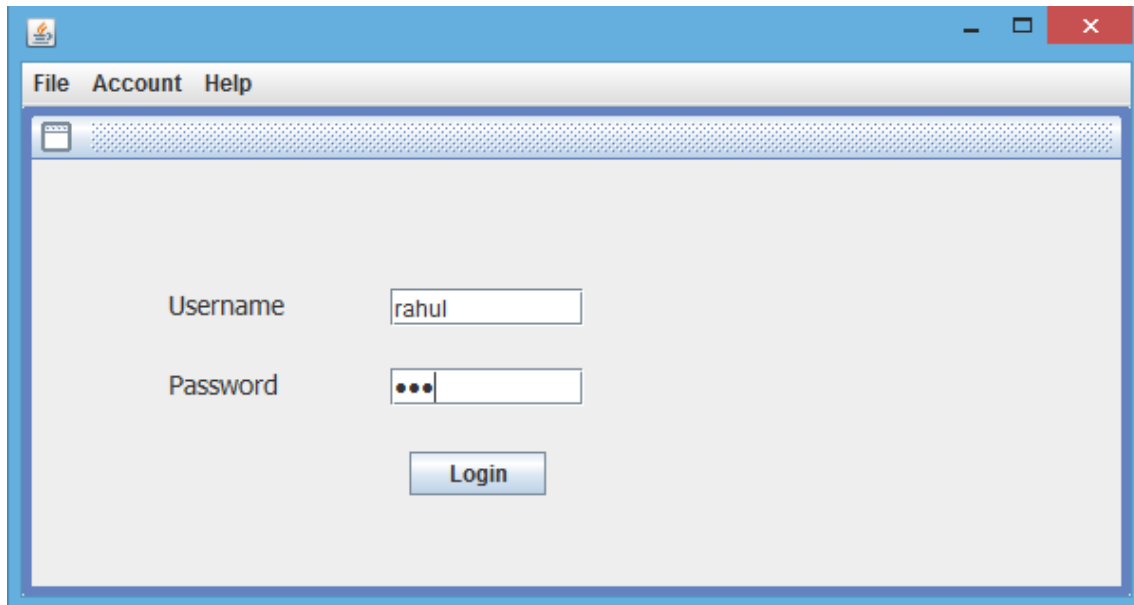


Figure 4.14: Login for Jingle Application

2. Test Scenario:

The users who can successfully log into the system should be able to see their buddy list and can transfer files to them successfully.

Test Cases:

No.	Function being Tested	Initial System State	Input	Expected Output
i.	Users can attach a file to the message which is to be sent to buddies cross-domain	Application shows Buddy List and option to 'Send File'	User clicks the 'Send File' button of the buddy with whom he wants to communicate	Application shows a dialog box to attach a file stored at the local system by navigating to the specified location using 'Browse' button
ii.	User is notified when the buddy receives the file	Application shows the location of file and 'OK' Button to send the file	User clicks the 'OK' button to send file	Application notifies the user when the file is successfully received by another user

Screenshots:

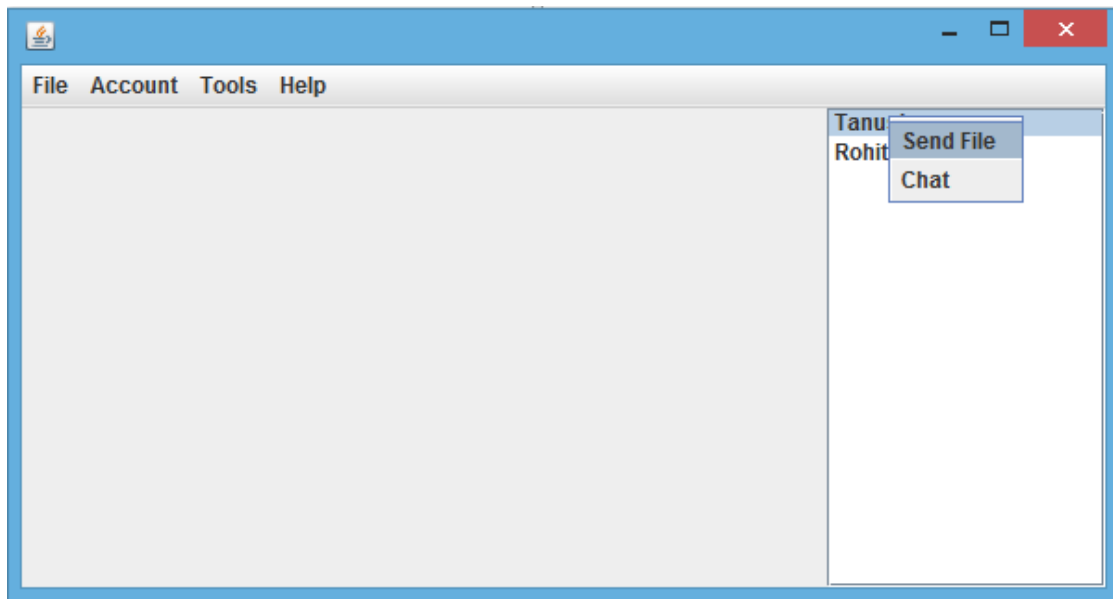


Figure 4.15: Buddy List in Jingle Application

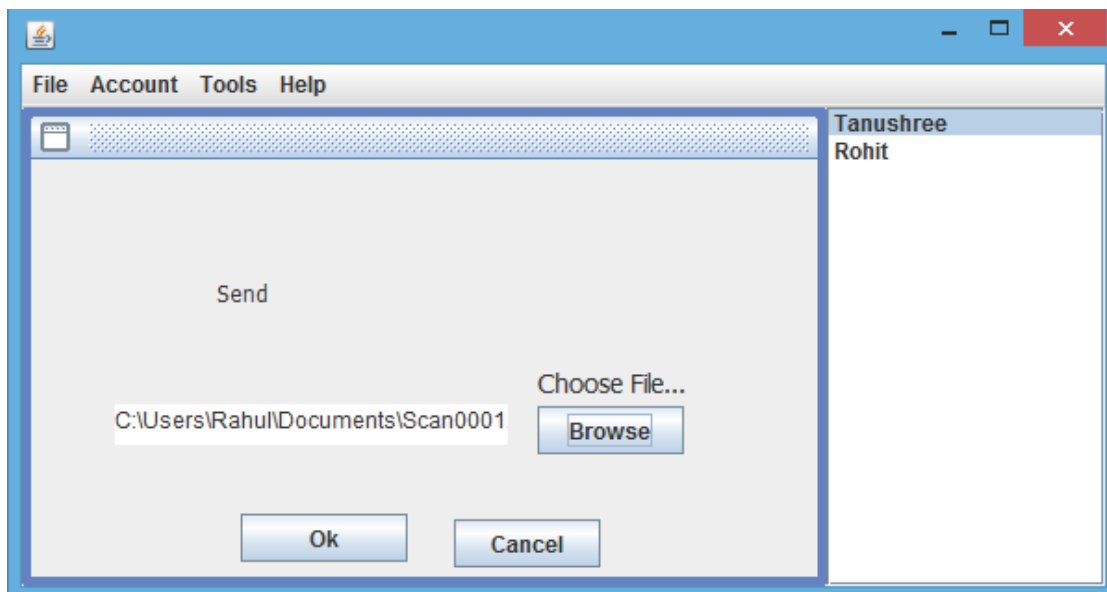


Figure 4.16: Send File in Jingle Application

3. Test Scenario :

If any buddy has sent file to the user, then the user should have the option to accept or reject the incoming file. If accepted, the file is stored at user's local storage.

Test Cases:

No .	Function being Tested	Initial System State	Input	Expected Output
	Users can approve or deny incoming files from his buddies	Application shows a dialog box with two buttons to accept and reject the incoming file	User clicks the 'OK' button to accept incoming file	Application receives the file from the buddy and stores it at the local storage path specified by user

Screenshot:

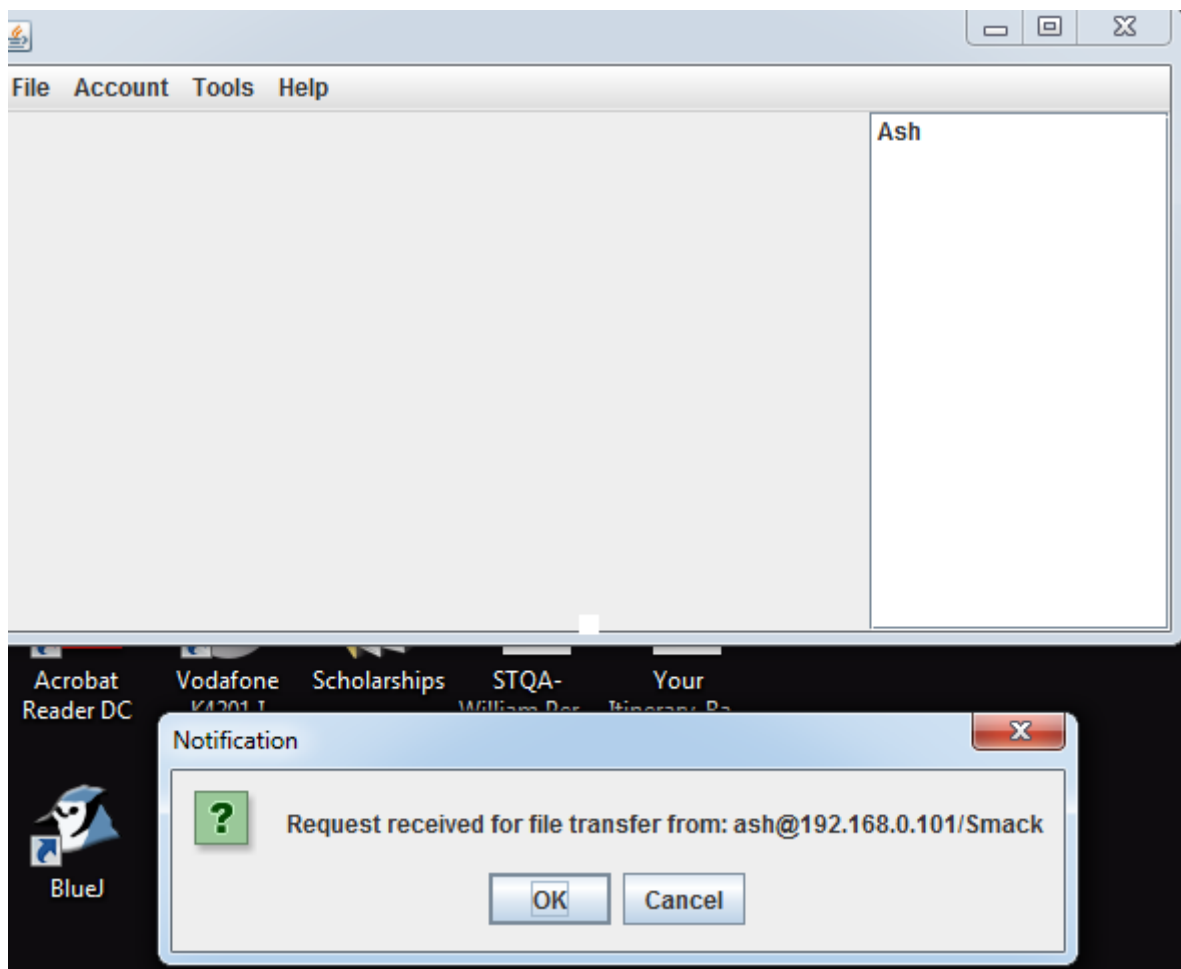


Figure 4.17: Request for incoming file

Chapter 5

Summary and Conclusions

Cross domain Communication allows users to create a group where in members can be using different servers and enables group communication between them. The XMPP protocol is the basic underlying protocol to achieve cross domain communication. It is an Internet Standard for real time messaging communication and presence using XML encoded protocol directly over TCP. To incorporate multimedia cross domain communication, we require SIP and Jingle along with XMPP Protocol. The Session Initiation Protocol (SIP) is a communication protocol for signalling and controlling multimedia communication sessions. Jingle sessions can support a wide range of application types (such as voice chat, video chat, and file transfer) and use a wide range of media transport methods (such as TCP, UDP, RTP, or even in-band XMPP itself). So, the best features of all the protocols mentioned above will be used to develop a new client for achieving audio communication at the cross domain level.

We have worked around with Openfire and AsteriskNow servers and studied single domain communication using existing chat clients. We intend to create a platform for cross domain communication.

Also, a need for providing this cross domain communication platform accessible to all the users from anywhere was realized and hence a web interface is required for the same. Hence, HTTP will be used along with the other protocols for achieving ubiquitous nature of the application. BOSH (Bidirectional streams Over Synchronous HTTP) is the standardized way to do XMPP over HTTP. BOSH defines a simple framing to use XML XMPP packets in HTTP protocol. As JavaScript is supported by all Web Browsers, a Web XMPP client needs to be written in JavaScript. Strophe.js is an XMPP library for JavaScript. Its primary purpose is to enable web-based, real-time XMPP applications that run in any browser.

A desktop application for audio is also developed where the extension to XMPP i.e. Jingle is used. Jingle uses the Session Initiation Protocol for session establishment and termination. Files transfer is done after the session is established. The Smack API for Java is used for connection establishment to Openfire server. The desktop application also provides features to transfer all other types of files.

Chapter 6

Appendix

6.1 Technical Reference Manual

Technologies needed:

1. Openfire server version 3.10.2
2. XAMPP or WAMP server having Apache and MySQL modules
3. Web browser: Chrome or Firefox
4. .war file for the web application
5. .jar file for the audio desktop application

6.1.1 Openfire server version 3.10.2

Openfire installation and deployment on system

Running Openfire in Windows:

If you used the Openfire installer, a shortcut for starting the graphical launcher is provided in your Start Menu. Otherwise, run `openfire.exe` in the **bin/** directory of your Openfire installation. A button on the on the launcher allows you to automatically open your web browser to the correct URL to finish setting up the server:

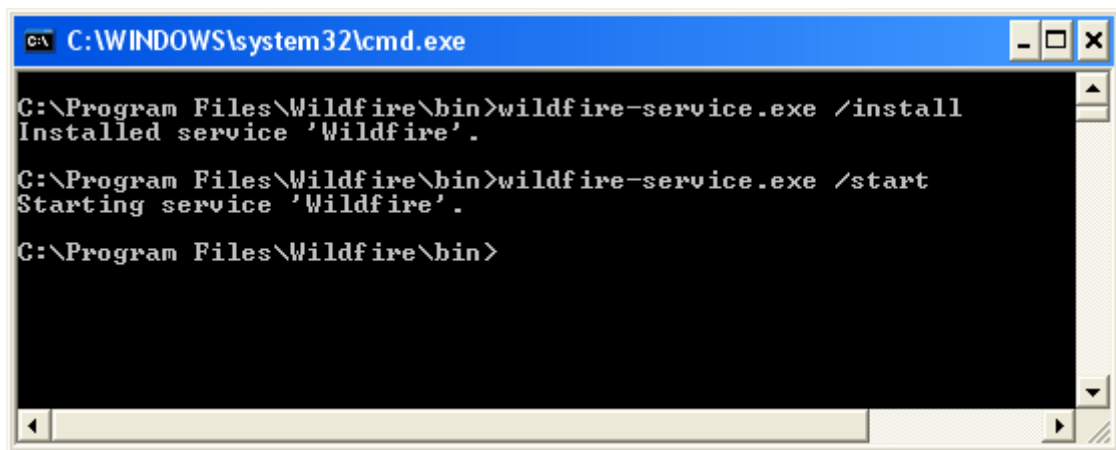


Windows Service:

If you're running Openfire on Windows, you will likely want to run Openfire as a standard Windows service after initial setup. If you used the Windows installer, a **openfire-service.exe** file will be in the **bin** directory of the installation. You can use this executable to install and control the Openfire service.

From a console window, you can run the following commands:

- **openfire-service /install** -- installs the service.
- **openfire-service /uninstall** -- uninstalls the service.
- **openfire-service /start** -- starts the service
- **openfire-service /stop** -- stops the service.



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Wildfire\bin>wildfire-service.exe /install
Installed service 'Wildfire'.

C:\Program Files\Wildfire\bin>wildfire-service.exe /start
Starting service 'Wildfire'.

C:\Program Files\Wildfire\bin>
```

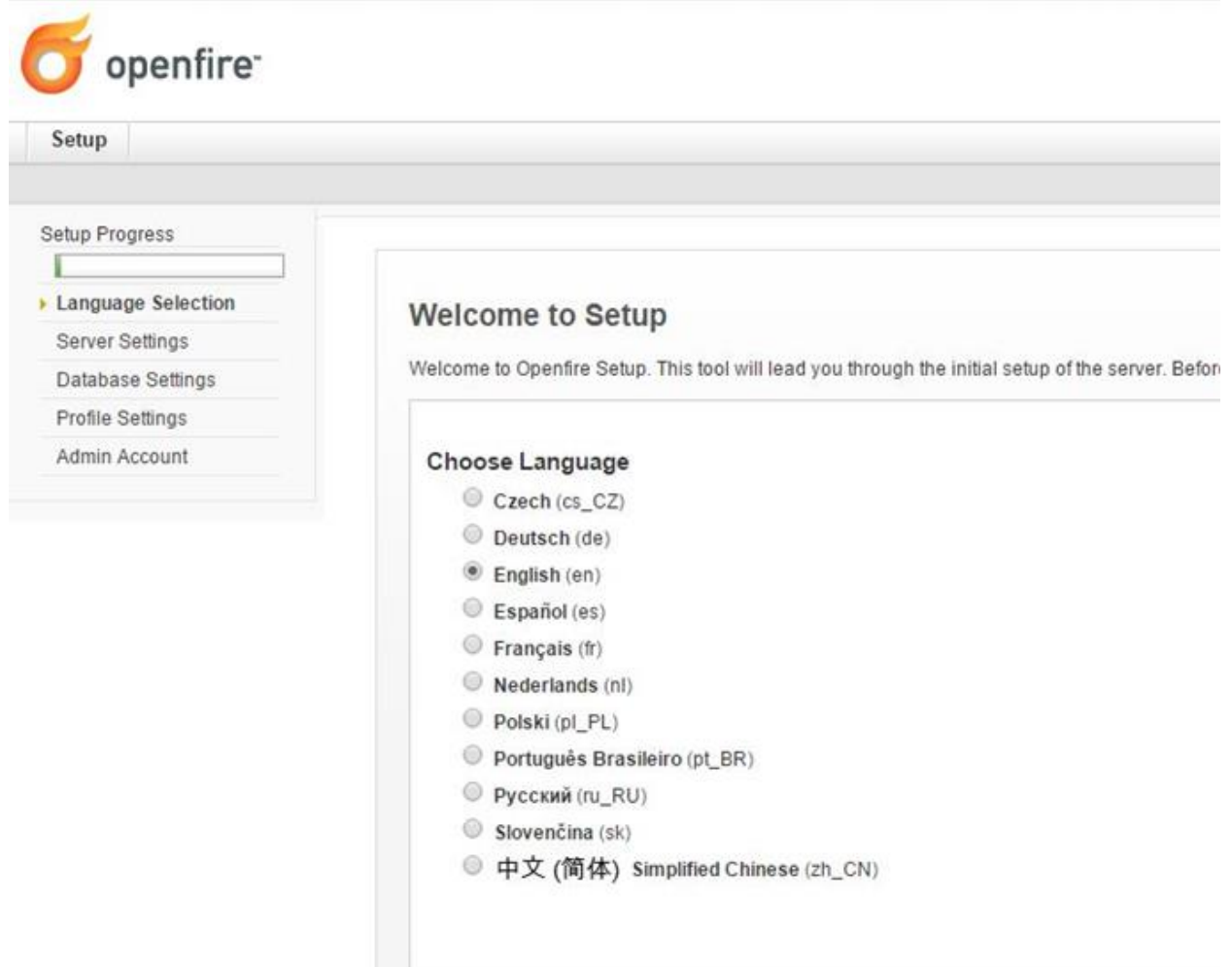
You can also use the Services tool in the Windows Control Panel to start and stop the service.

We have the following 5 steps to complete the configuration:

1. Language Selection
2. Server Settings
3. Database Settings
4. Profile Settings
5. Admin Accounts

Step 1: Language Selection

Choose the language which you want the installer and the admin console to be in and then click on 'Continue'.



Step 2: Server Settings

In this step we configure the server domain, admin console port, and also the secure admin port. Generally, we do not change these data, unless and until we need custom port. But make sure that these ports are open on your network.

Setup

Setup Progress

✓ Language Selection

▶ **Server Settings**

Database Settings

Profile Settings

Admin Account

Server Settings

Below are host settings for this server. Note: the suggested value for the domain is based on the netv

Domain:

?

Admin Console Port:

9090

?

Secure Admin Console Port:

9091

?

Property Encryption via:

?

☒ Blowfish

☐ AES

Property Encryption Key:

?

Step 3: Database Settings

Here we have 2 ways to setup the database:

Embedded Database

Standard Database Connection

Embedded Database: Embedded database is the easiest way to configure and setup as it does not require any external database. However, it does not give the same level of performance as given by an external database.

If you choose to use Embedded Database and click continue then it will take you to Profile settings.

Standard Database: Standard Database connection requires an external database up and running.

By default Standard Database is selected, click on continue to configure to external database.

Openfire Setup: Database x Sagar Panda

127.0.0.1:9090/setup/setup-datasource-standard.jsp

openfire

Setup

Setup Progress

- ✓ Language Selection
- ✓ Server Settings
- Database Settings
- Profile Settings
- Admin Account

Database Settings - Standard Connection

Specify a JDBC driver and connection properties to connect to your database. If you need more information about this process please see the database documentation distributed with Openfire.

Note: Database scripts for most popular databases are included in the server distribution at `(Openfire_HOME)/resources/database`.

Database Driver Presets:

JDBC Driver Class:

Database URL:

Username:

Password:

Minimum Connections:

Maximum Connections:

Connection Timeout: Days

Note, it might take between 30-60 seconds to connect to your database.

[Continue](#)

Database URL: jdbc:mysql://[host-name]/[database-name]?rewriteBatchedStatements=true
[host-name] = IP address of your server
[database-name] = name of the database to which you wish to configure with openfire
Username/Password = username and password of the login you created for your database.

Step 4: Profile Settings

It allows you to choose methods for storing and authenticating users and group data.

For more simplicity, I rely on default option and continue.

Openfire Setup: Profile Se x

127.0.0.1:9090/setup/setup-profile-settings.jsp

openfire

Setup

Setup Progress

- ✓ Language Selection
- ✓ Server Settings
- ✓ Database Settings
- Profile Settings
- Admin Account

Profile Settings

Choose the user and group system to use with the server.

- ☒ **Default**
Store users and groups in the server database. This is the best option for simple deployments.
- ☐ **Directory Server (LDAP)**
Integrate with a directory server such as Active Directory or OpenLDAP using the LDAP protocol.
- ☐ **Clearspace Integration**
Integrate with an existing Clearspace installation. Users and groups will be pulled directly from C Clearspace 2.0 or higher is required.

Step 5: Admin Account

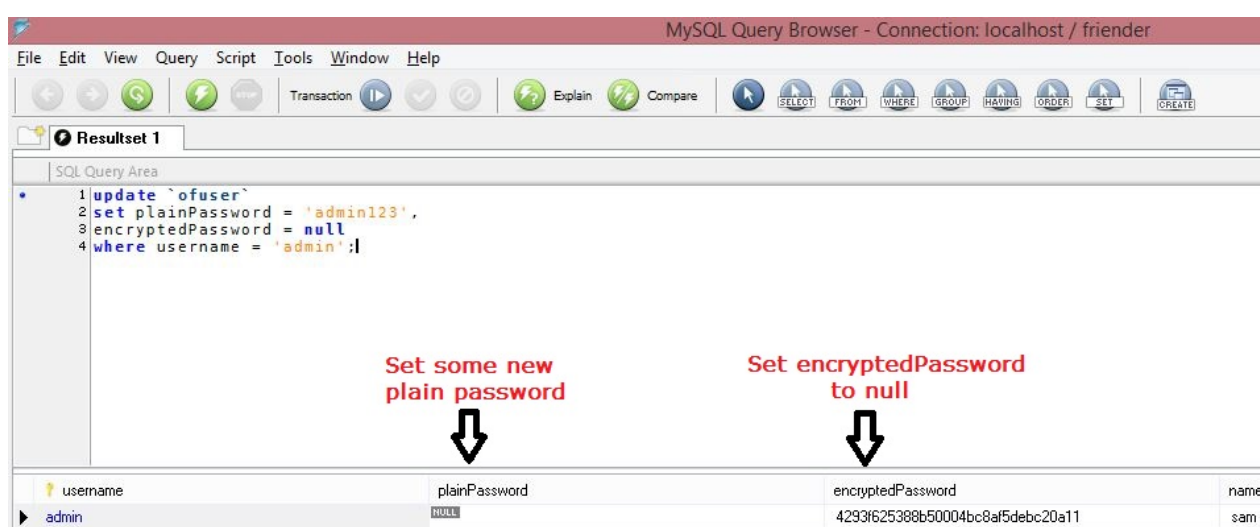
This is the final step of our web base configuration. Make sure you enter a valid email address for your admin. The default password for admin is 'admin'. Enter your new password and confirm password and continue to proceed to see the Setup Complete message.

Now, if we try to login to the admin console, it will give the following error: **“Login failed: make sure your username and password are correct and that you are an admin or moderator”** Even though your credentials are correct, it won’t allow us to login.

Go to the openfire server database,(this is the external standard database which we configured in Step 3)

Check the of User table which contains the list of users that we register/add on our openfire.

Here we need to change the password of admin by simply firing the below query:



The screenshot shows the MySQL Query Browser interface. The SQL Query Area contains the following query:

```
1 update `ofuser`  
2 set plainPassword = 'admin123',  
3 encryptedPassword = null  
4 where username = 'admin';
```

Below the query, there are two red annotations with arrows pointing to the table view:

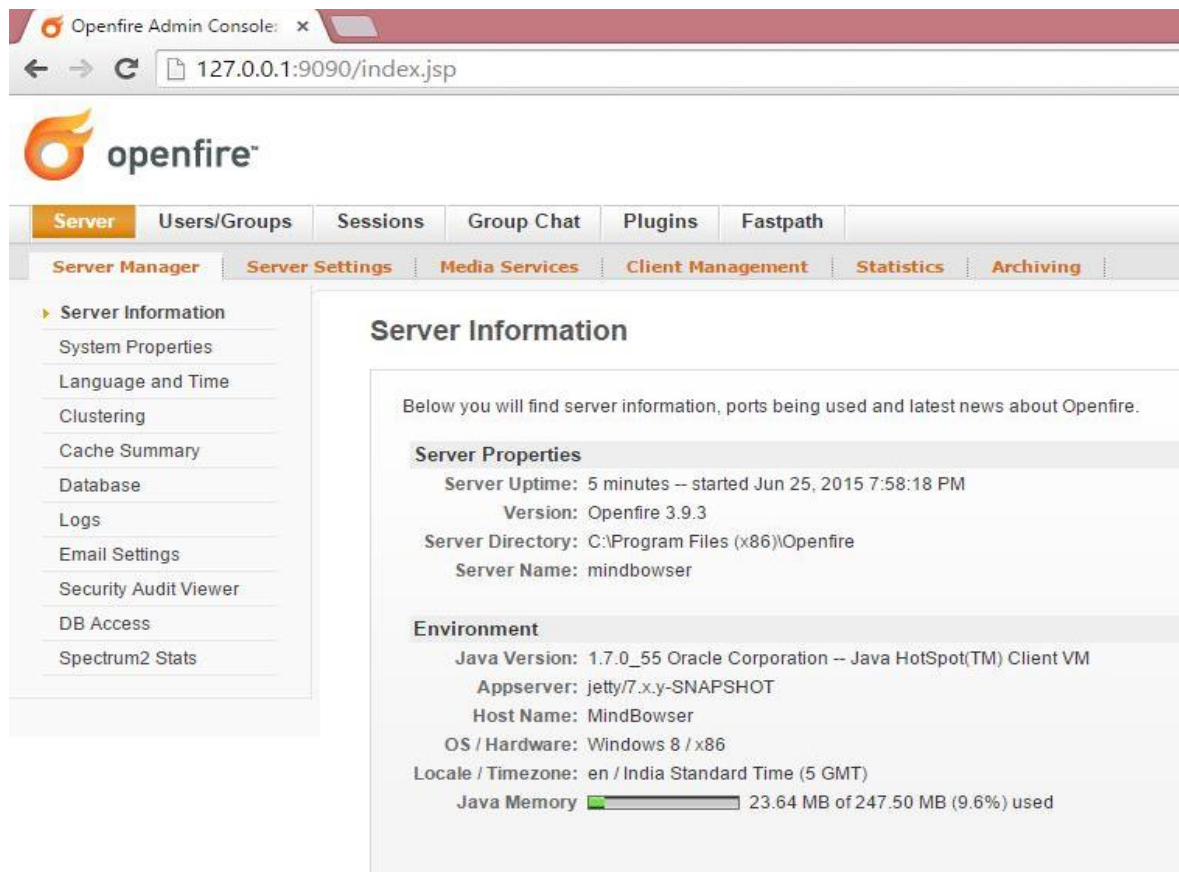
- Set some new plain password** (arrow pointing to the 'plainPassword' column)
- Set encryptedPassword to null** (arrow pointing to the 'encryptedPassword' column)

The table view shows the following data:

username	plainPassword	encryptedPassword	name
admin	NULL	4293f625388b50004bc8af5debc20a11	sam

You can set plain Password of your own choice.

Now you can login to the admin console (<http://127.0.0.1:9090/>) using the username as admin and password as this new plain password.



Once the setup is completely done, you can check your database to which we have linked openfire. You can see that new tables have been added to your database by the openfire. For a complete list of database tables and schema you can read [here](#).

Openfire Admin Console:

You have an entire control on your openfire server through the admin console. Let's look at some of important and most used properties and functionalities which can be achieved through admin console.

Creating a user:

In the admin console:

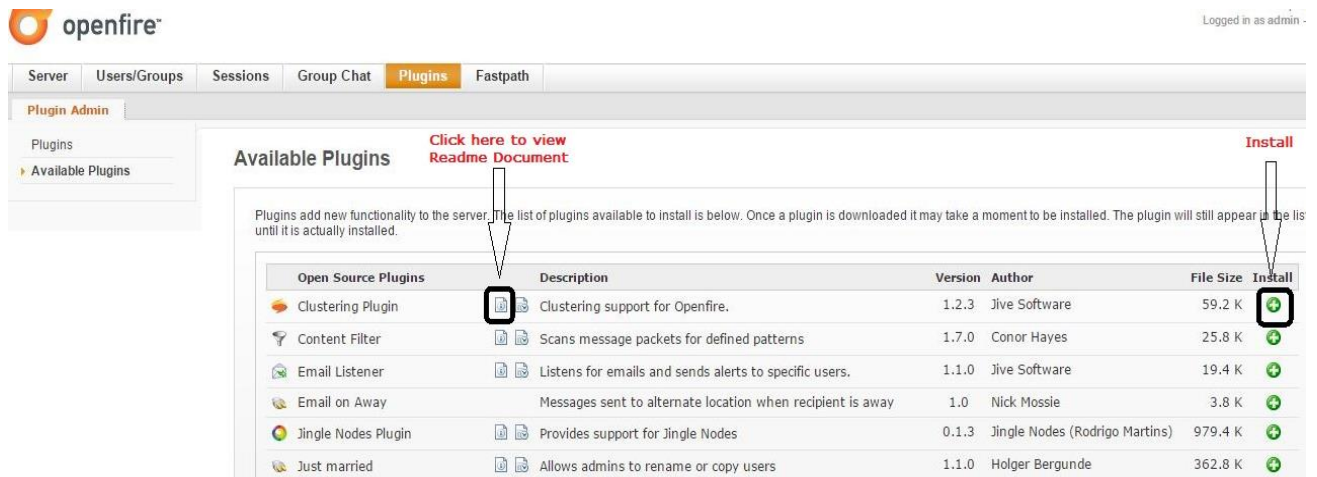
1. Click on **User/Groups**
2. Under Users section click on **Create New User**
3. Fill all the required details and click **Create User**

The screenshot shows the Openfire Admin Console interface in a web browser. The browser's address bar displays the URL `127.0.0.1:9090/user-create.jsp`. The Openfire logo is visible at the top left. The main navigation bar includes tabs for **Server**, **Users/Groups** (which is selected), **Sessions**, **Group Chat**, **Plugins**, and **Fastpath**. Below this, a sub-navigation bar shows **Users**, **Groups**, and **Import & Export**. On the left side, a sidebar menu lists several options: **User Summary**, **Create New User** (highlighted with a yellow arrow), **User Search**, **Registration Properties**, **Advanced User Search**, and **Users Creation**. The main content area is titled **Create User** and contains the instruction: "Use the form below to create a new user." Below this instruction is a form titled **Create New User** with the following fields: **Username: *** (a text input field), **Name:** (a text input field), **Email:** (a text input field), **Password: *** (a text input field), **Confirm Password: *** (a text input field), and **Is Administrator?** (a checkbox followed by the text "(Grants admin access to Openfire)"). At the bottom of the form are three buttons: **Create User**, **Create & Create Another**, and **Cancel**. A legend at the bottom left of the form indicates that an asterisk (*) denotes required fields.

Then, under **User Summary** you can see the newly created user.

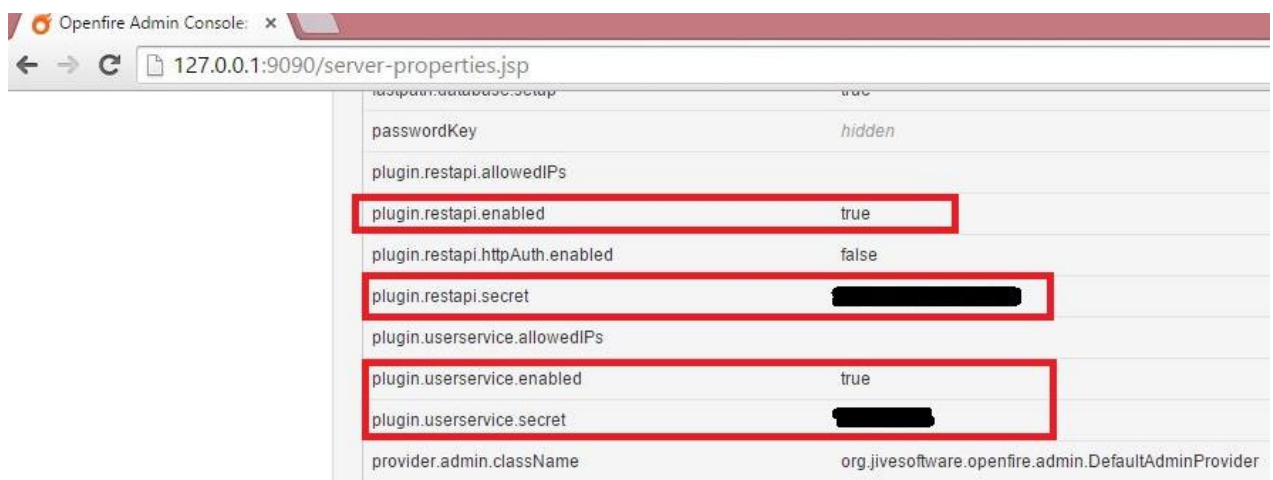
This was one way of creating user through the admin console. Openfire has provided us with the functionality of REST api's for various operations like creating/deleting/modifying user, adding/deleting a user to/from a roster of another user and many other operations can be performed easily by the hit of a url with some parameters added to it.

For this you will require functionality specific **PLUGINS** to be installed in the openfire server. Every plugin has its own functionality which has been mentioned clearly in its readme document.



You can install plugins by simply clicking on the install button. Once installed, restart the openfire server. You can also see jar files of the installed plugins in the installation directory of openfire server, generally which is C:\Program Files (x86)\Openfire\plugins.

While using REST or User Service we will require the secret keys for security purpose, which you can find it under Server ⇒ Server Manager ⇒ System Properties.



System properties are an aggregate of all the properties and settings of the openfire. Meanwhile you can also check and edit any specific properties under Server ⇒ Server Settings.

6.1.2 XAMPP or WAMP server having Apache and MySQL modules

XAMPP stands for Cross-Platform (X), Apache (A), MySQL (M), PHP (P) and Perl (P). It is a simple, lightweight Apache distribution that makes it extremely easy for developers to create a local web server for testing purposes. Everything you need to set up a web server – server application (Apache), database (MySQL), and scripting language (PHP) – is included in a simple extractable file. XAMPP is also cross-platform, which means it works equally well on Linux, Mac and Windows. Since most actual web server deployments use the same components as XAMPP, it makes transitioning from a local test server to a live server is extremely easy as well.

Installing XAMPP

Follow these steps for installing XAMPP:

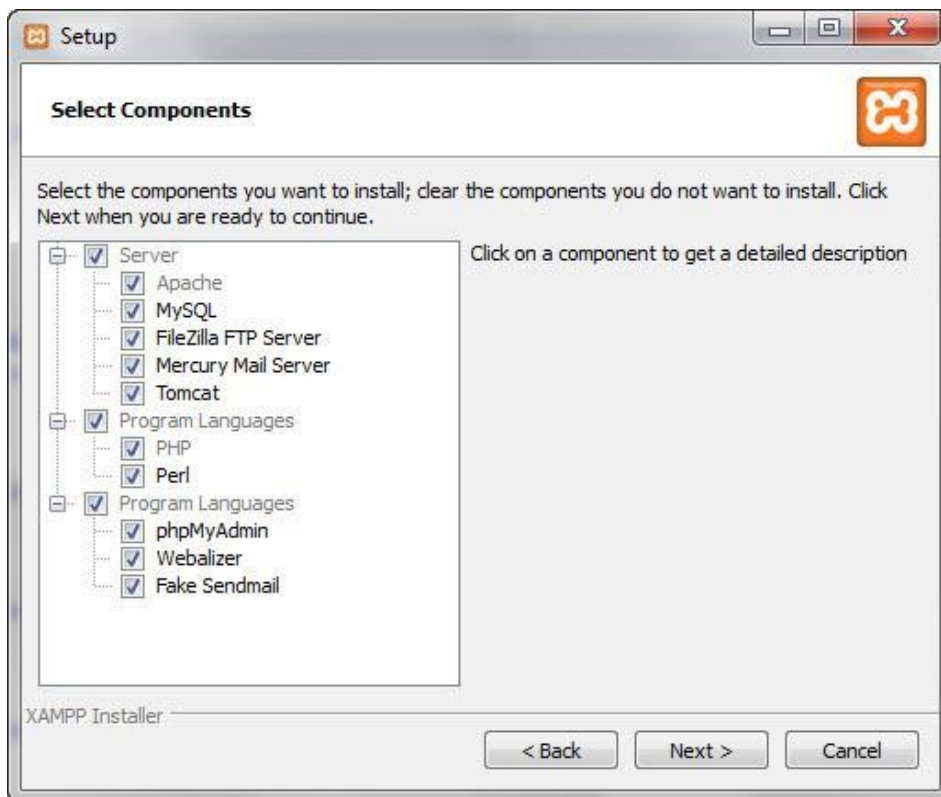
Step 1: Disable your anti-virus as it can cause some XAMPP components to behave erratically.

Step 2: Disable User Account Control (UAC). UAC limits write permissions to XAMPP's default installation directory (c:/Program Files/xampp), forcing you to install in a separate directory. You can learn how to disable UAC [here](#). (Optional)

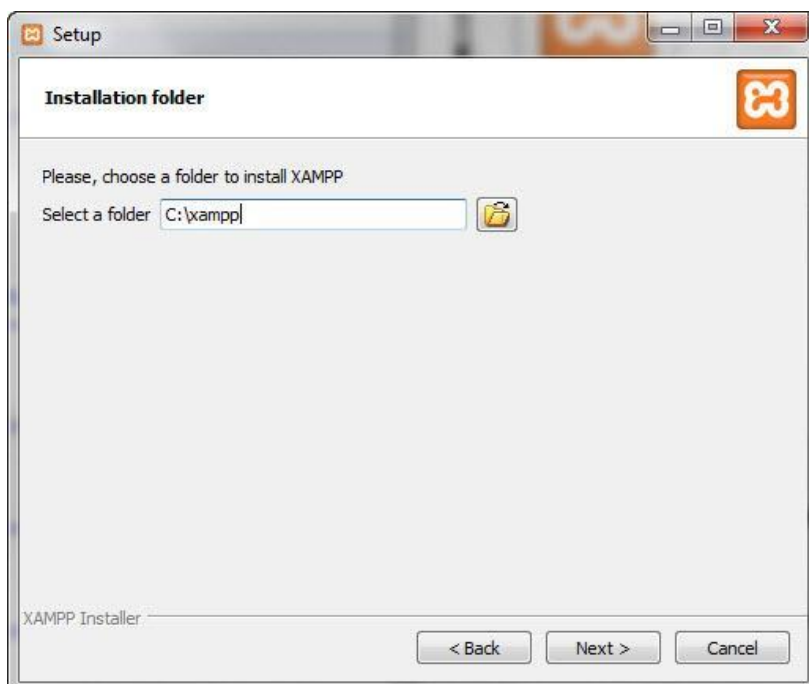
Step 3: Start the installation process by double-clicking on the XAMPP installer. Click 'Next' after the splash screen.



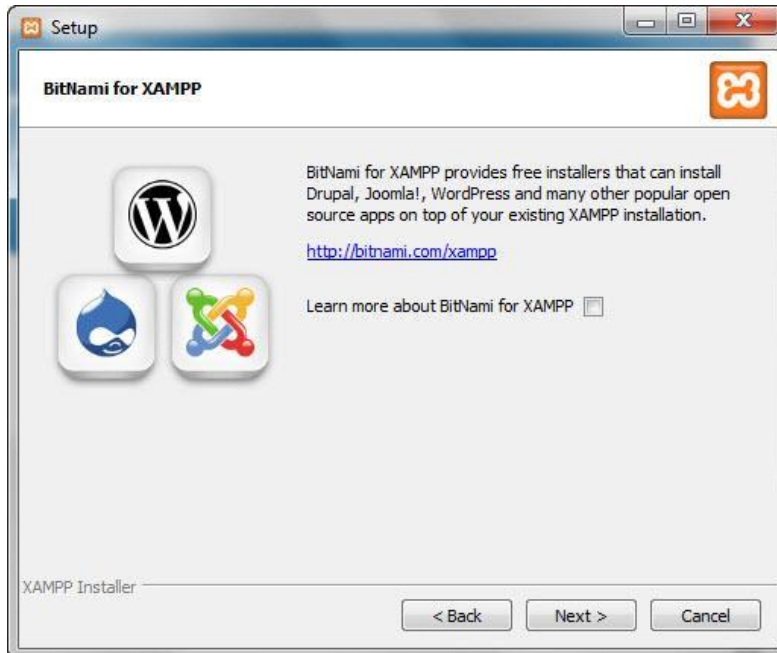
Step 4: Here, you can select the components you want to install. Choose the default selection and click 'Next'.



Step 5: Choose the folder you want to install XAMPP in. This folder will hold all your web application files, so make sure to select a drive that has plenty of space.



Step 6: The next screen is a promo for BitNami, an app store for server software. Deselect the ‘Learn more about BitNami for XAMPP’ checkbox, unless you actually enjoy receiving promo mails!



Step 7: Setup is now ready to install XAMPP. Click Next and wait for the installer to unpack and install selected components. This may take a few minutes. You may be asked to approve Firewall access to certain components (such as Apache) during the installation process.

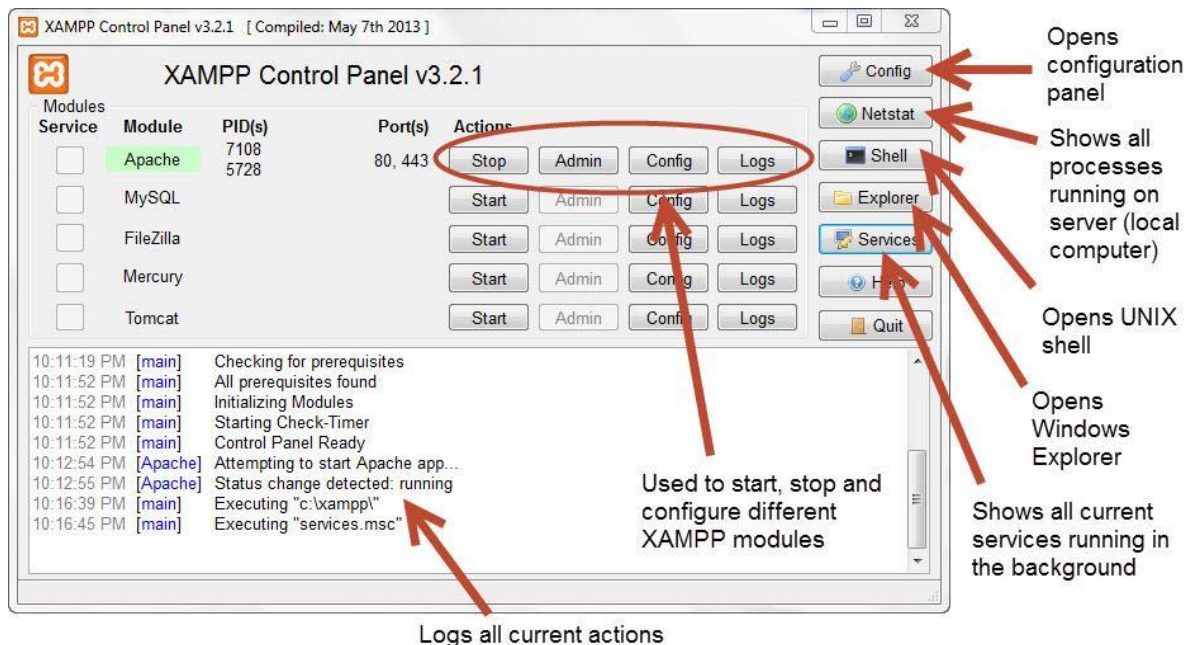
Step 8: Installation is now complete! Select the ‘Do you want to start the Control Panel now?’ checkbox to open the XAMPP control panel.



Understanding XAMPP Control Panel

The XAMPP control panel gives you complete control over all installed XAMPP components. You can use the CP to start/stop different modules, launch the Unix shell, open Windows explorer and see all operations running in the background.

Here is a quick overview of the Control Panel. For now, you only need to know how to start and stop an Apache server.



Testing Your XAMPP Installation

Follow these steps to test your XAMPP installation by launching the Apache web server and creating a simple PHP file.

Step 1: In the XAMPP control panel, click on 'Start' under 'Actions' for the Apache module. This instructs XAMPP to start the Apache webserver.

Step 2: Open your web browser and type in: <http://localhost> or 127.0.0.1

Step 3: Select your language from the splash screen.



[English](#) / [Deutsch](#) / [Français](#) / [Nederlands](#) / [Polski](#) / [Italiano](#) / [Norwegian](#) / [Español](#) / [中文](#) / [Português \(Brasil\)](#) / [日本語](#)

Step 4: You should see the following screen. This means you've successfully installed XAMPP on your computer.



Thus, we have successfully set up a Xampp local Server on the local machine.

6.1.3 Web browser: Chrome or Firefox

As OpenFire Server is best compatible with Google Chrome and the default browser in the configuration files of OpenFire is Google Chrome, so it is advisable to use Google Chrome. To use Firefox, additional changes will be required to be done in the configuration file.

6.1.4 War file for the web application

A war (web archive) File contains files of a web project. It may have servlet, xml, jsp, image, html, css, js etc. files.

Creating war file:

To create war file, you need to use jar tool of JDK. You need to use -c switch of jar, to create the war file.

Go inside the project directory of your project (outside the WEB-INF), then write the following command:

```
jar -cvf projectname.war *
```

Here, -c is used to create file, -v to generate the verbose output and -f to specify the archive file name.

The * (asterisk) symbol signifies that all the files of this directory (including sub directory).

Deploying war file:

There are two ways to deploy the war file.

1. By server console panel
2. By manually having the war file in specific folder of server.

If you want to deploy the war file in apache tomcat server manually, go to the webapps directory of apache tomcat and paste the war file here.

Now, you are able to access the web project through browser.

6.1.5 Jar file for the audio desktop application

In Java, it is common to combine several classes in one .jar ("java archive") file. Library classes are stored that way. Larger projects use jar files. You can create your own jar file combining several classes, too. jar files are created using the jar.exe utility program from the JDK. You can make your jar file runnable by telling jar.exe which class has main. To do that, you need to create a manifest file. A manifest is a one-line text file with a "Main-Class" directive.

Creating a JAR File

The basic format of the command for creating a JAR file is:

```
jarcfjar-file input-file(s)
```

The options and arguments used in this command are:

- The *c* option indicates that you want to *create* a JAR file.
- The *f* option indicates that you want the output to go to a *file* rather than to stdout.
- *jar-file* is the name that you want the resulting JAR file to have. You can use any filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required.
- The *input-file(s)* argument is a space-separated list of one or more files that you want to include in your JAR file. The *input-file(s)* argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

The *c* and *f* options can appear in either order, but there must not be any space between them. This command will generate a compressed JAR file and place it in the current directory. The command will also generate a default manifest file for the JAR archive.

6.2 User Manual

Web application for Text based cross-domain communication:

1. Users need a working internet connection.
2. They need an XMPP account. The account is created at the server side.
3. Users have to login at the url provided using the XMPP server (Openfire server) account username and password.
4. If the username and password are correct, the user is successfully authenticated.
Otherwise, connection to server is refused and all the functionalities are disabled.
5. On successful login, the user can now retrieve his roster i.e buddy list.
6. The user can add contacts in his buddy list using the Add Contact button. He has to provide the jid and name of the contact to be added. Then a subscription request gets sent to the contact.
7. The user may also receive a subscription request from other users. He has a choice to accept or reject the subscription.
8. The user can chat with contacts in his roster. He opens a new chat tab by clicking on the contact.
9. He can also chat with contacts not in his contact list.
10. The user also has options to log out of the XMPP Web Chat by clicking on the Disconnect button.

Desktop Application for Audio communication:

1. The desktop application is used by users to perform audio file transfer over XMPP protocol.
2. For performing audio communication, the user has to authenticate himself into the system by providing username and password.
3. Then the user's roster list is displayed which includes the contacts in his roster.
4. The user then selects the option to send file after which a file chooser is opened.
5. The recorded audio file is then chosen and sent to the contact.
6. The receiver of the file has an option to accept or reject the request for file transfer.
7. The user can log out by clicking on the Disconnect button.

Chapter 7

References

1. <http://oriolrius.cat/blog/wpcontent/uploads/2009/10/Oreilly.XMPP.The.Definitive.Guide.May.2009.pdf>
2. https://en.wikipedia.org/wiki/Session_Initiation_Protocol
3. <http://www.siptutorial.net/SIP/>
4. http://www.asteriskdocs.org/en/2nd_Edition/asterisk-book-html-chunk/asterisk-CHP-3-SECT-14.html
5. <http://www.asterisk.org/get-started>
6. https://en.wikipedia.org/wiki/Jingle_%28protocol%29
7. xmpp.org/extensions/xep-0167.html
8. <http://www.isode.com/whitepapers/xmpp-bosh.html>
9. <http://strophe.im/strophejs/doc/1.2.3/files/strophe-js.html>
10. <http://xmpp.org/rfcs/rfc6121.html>
11. <http://www.rfc-base.org/txt/rfc-3261.txt>
12. P. Saint-Andre, "Streaming XML with Jabber/XMPP", *IEEE Internet Comp.*, vol. 9, no. 5, pp. 82-89, 2005
13. Chengzhou Fu Sch. of Comput. Sci., South China Normal Univ., Guangzhou, China
Yong Tang ; Chengzhe Yuan ; Yuying Xu, "Cross-Platform Instant Messaging System", 2015
12th Web Information System and Application Conference (WISA)
14. N. Lass Robert , "XO: XMPP overlay service for distributed chat", MILITARY
COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010, IEEE

15. Multi User Chat (MUC) - XEP-0045." [Online]. Available: <http://xmpp.org/extensions/xep-0045.html>
16. B. A. Nardi, S. Whittaker and E. Bradner, "Interaction and Outeraction: Instant Messaging in Action", *Proc. ACM Conference Computer Supported Cooperative Work*, pp. 79-88 [CrossRef]
17. P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): Instant messaging and presence," Internet Engineering Task Force (IETF), Request for Comments 6121, March 2011.
18. P. Saint-Andre, K. Smith, and R. Tronçon, XMPP: The Definitive Guide. O'Reilly, April 2009.
19. "XEP-0045: Multi-User Chat," XMPP Standards Foundation, Standards Track, Dec. 2008. [Online]. Available: <http://xmpp.org/extensions/xep-0045.html>
20. G. Chen, "XMPP Pubsub Extension for Long-lived TCP Services," IETF, Internet-Draft, Jul. 2011. [Online]. Available: <http://tools.ietf.org/html/draft-chen-xmpp-pubsub-extension-00>
21. T. Eugster, A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many Faces of Publish/Subscribe," *ACM Comput. Surv.* 35 (2), 2003.