**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:

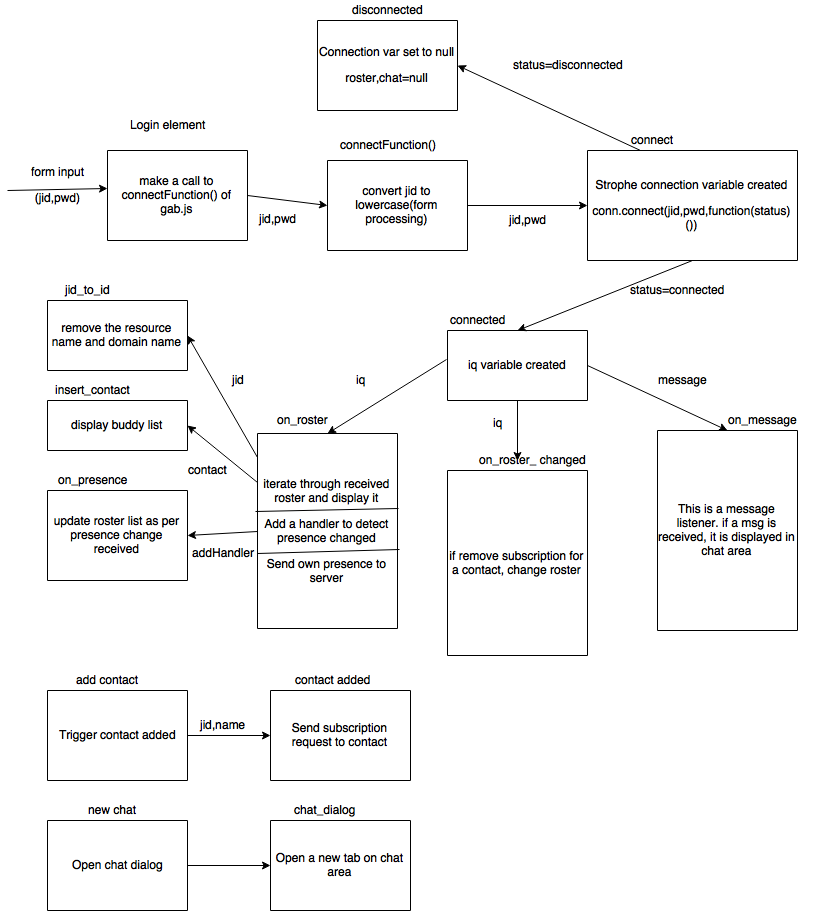


Figure 4.1: Component diagram for Web Application

**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** |
|  | 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 |
|  | 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:**

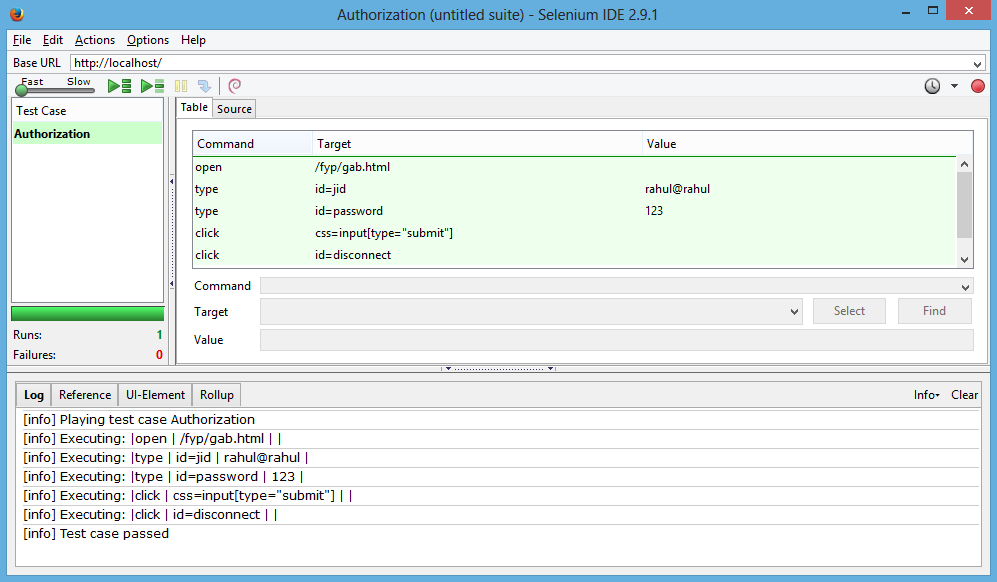
****

Figure 4.2: Authorization Test Case



Figure 4.3: Login Page

1. **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** |
|  | 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 |
|  | 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) |
|  | 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:**

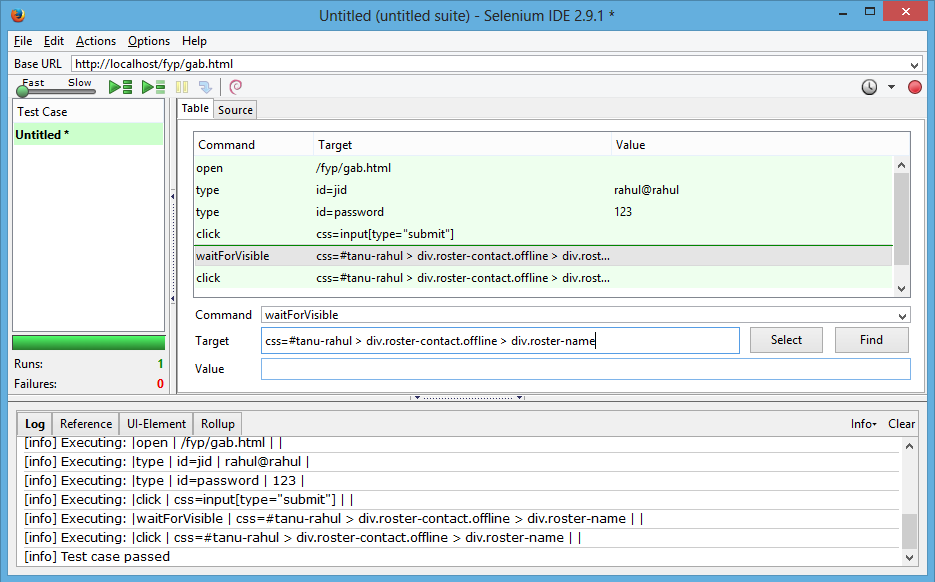


Figure 4.4: Buddy List Test Case

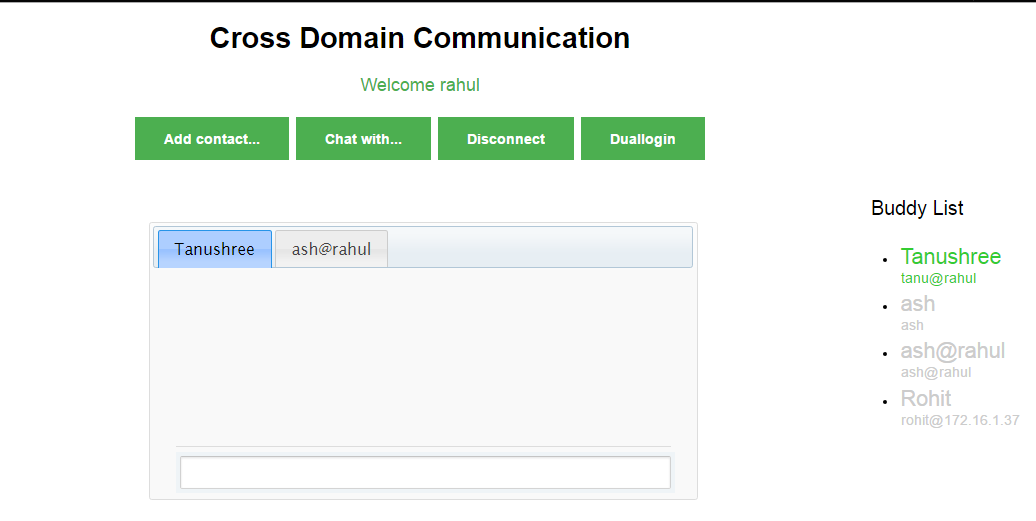


Figure 4.5: Buddy List

1. **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** |
|  | 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 |
|  | User can view his buddy’s activities | Application shows a chat area with the buddy name | No user input | Application shows that the buddy who wants to communicate with you is typing something |

**Screenshots:**

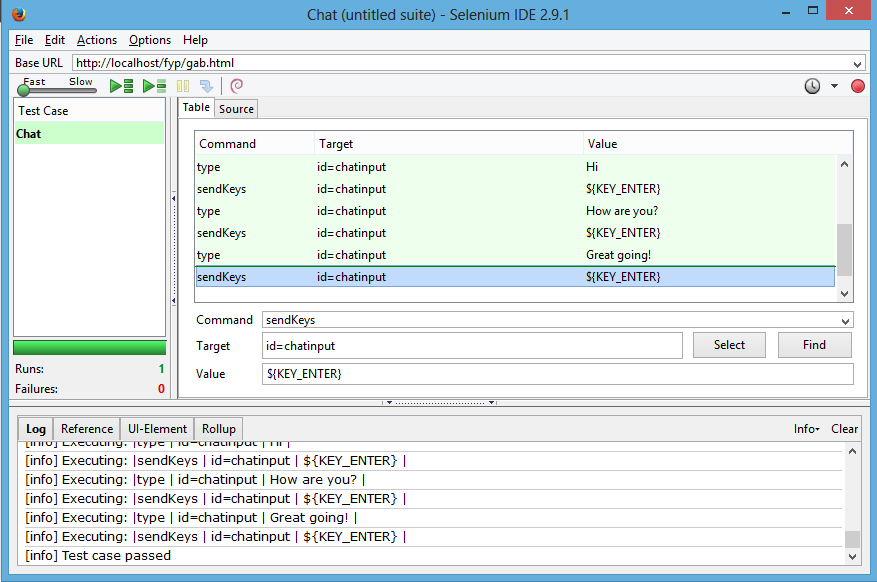
****

Figure4.6: Chat Test Case

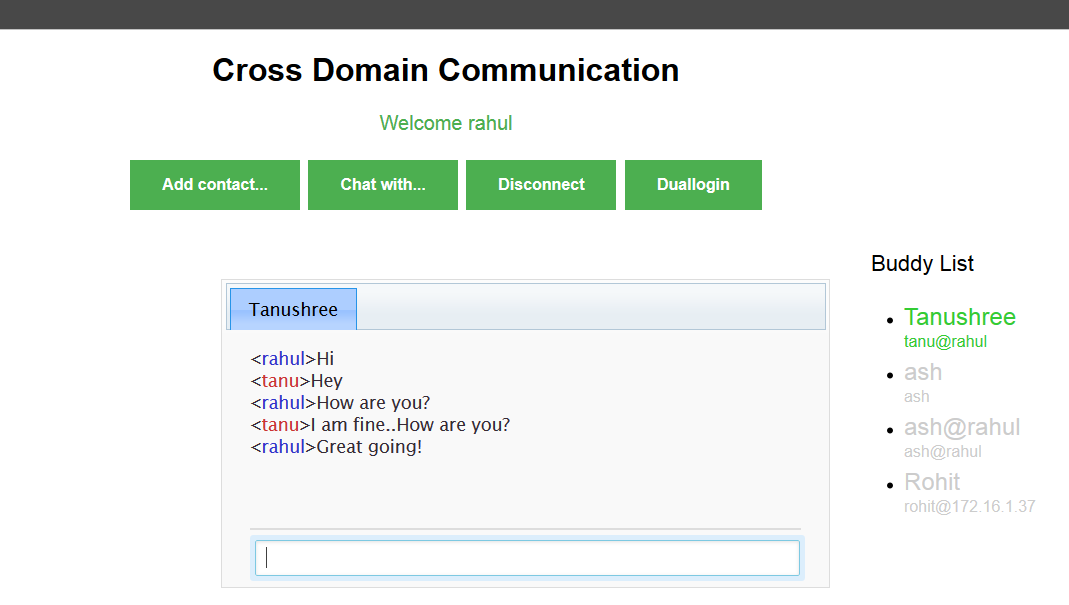
****

Figure 4.7: Chat using Single Domain

1. **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** |
|  | 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 to whom 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 |
|  | 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:**

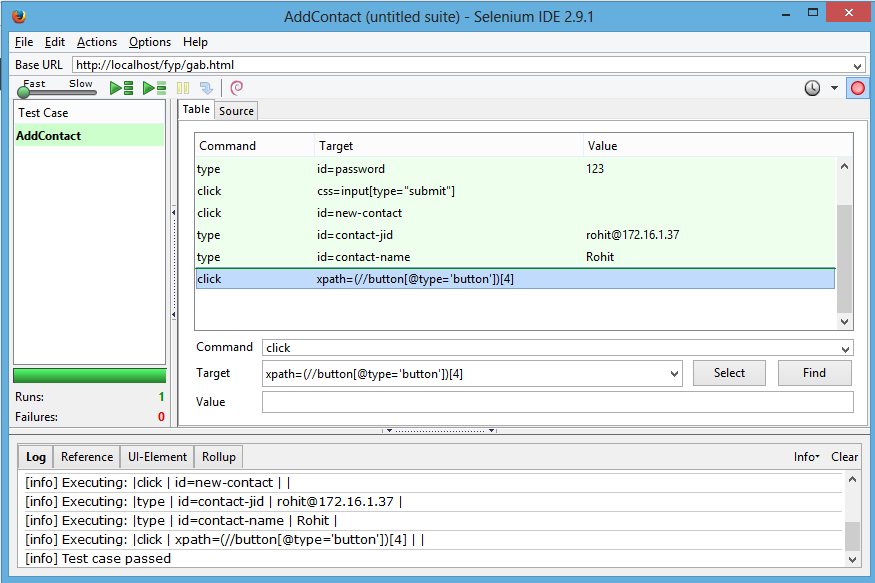
****

Figure 4.8: Add Contact Test Case

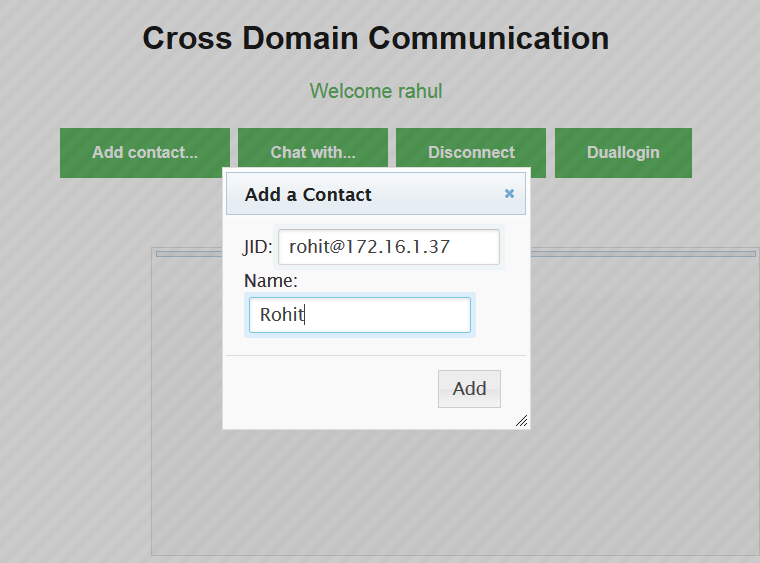
****

Figure 4.9: Add Contact from another domain

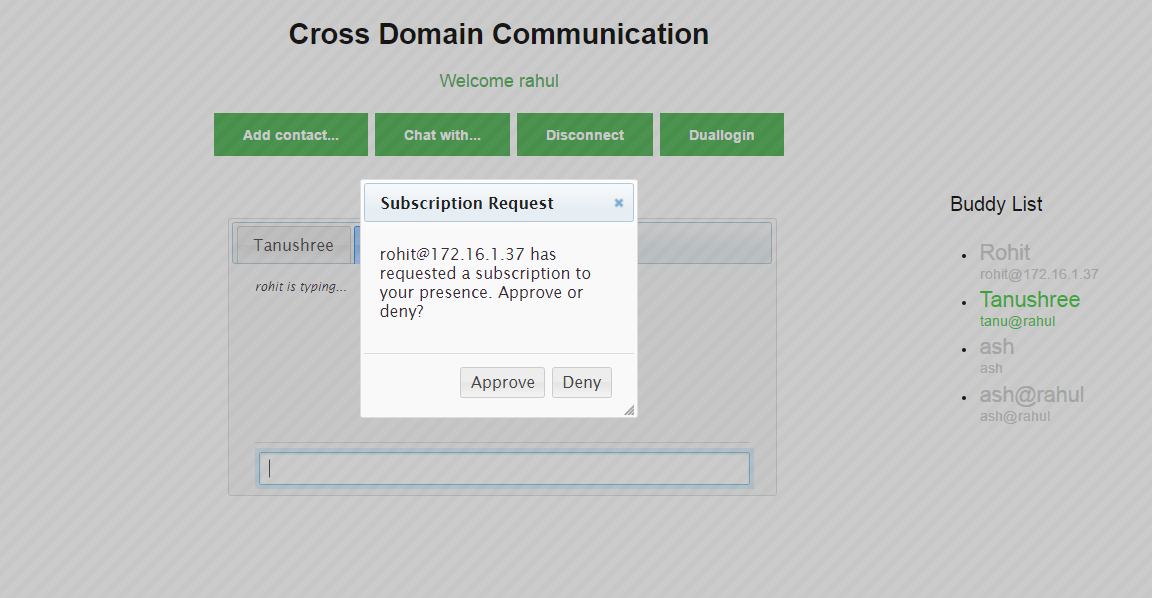


Figure 4.10: Subscription Request

1. **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** |
|  | 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:**

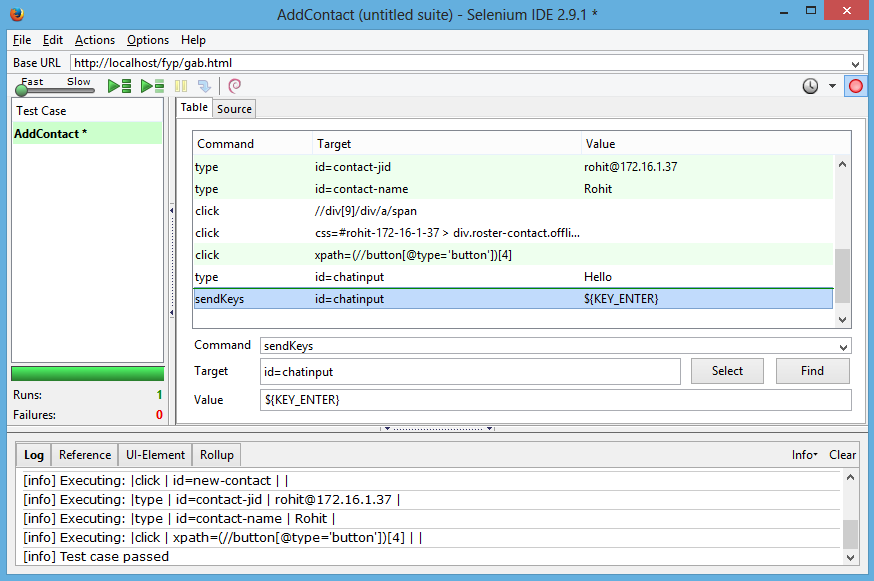
****

Figure 4.11: Chat with other domain user Test Case

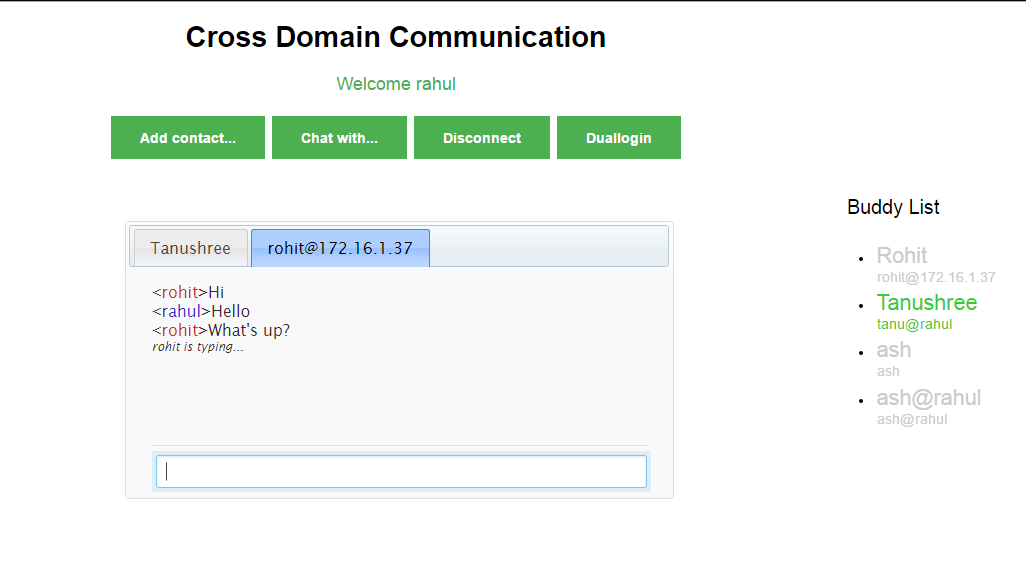
****

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:

**4.4 Software Testing: Jingle Application (Functional Test Cases)**

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** |
|  | 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 |
|  | 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:**

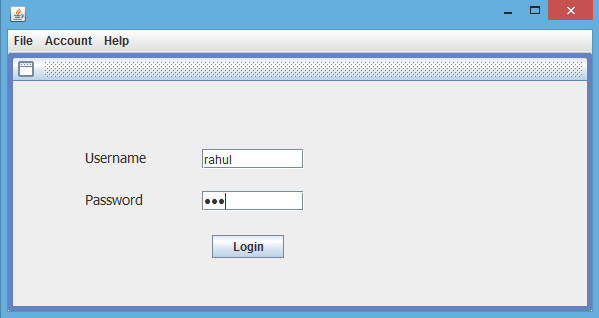
****

Figure 4.14: Login for Jingle Application

1. **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** |
|  | 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 |
|  | 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:**

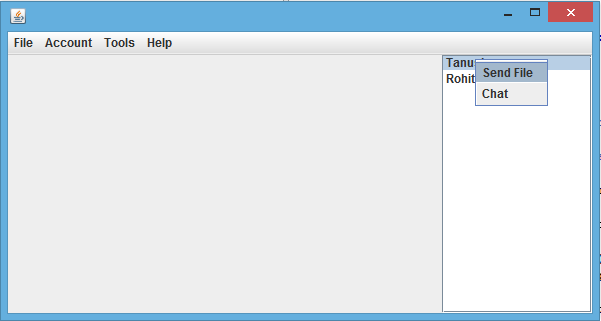
****

Figure 4.15: Buddy List in Jingle Application

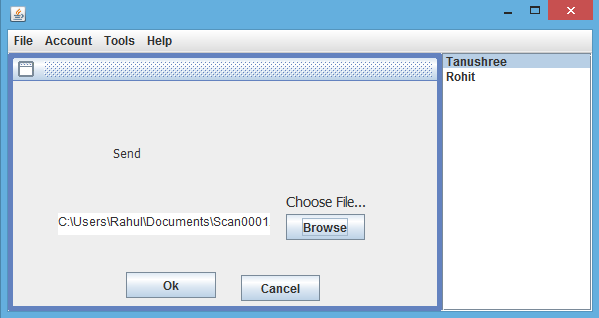
****

Figure 4.16: Send File in Jingle Application

1. **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 |

**Screenshots:**