Enable Private and Independent Communication Using XMPP Protocol

By: Daniel Roldan

Purpose

The purpose of this application is to allow for secure intranet communication within the same premises. Meaning that anyone within the same network can communicate freely through emails and/or instant message. All this is done without ever having to access the internet. The communication data stays within the boundaries of the network and all sensitive information stays within the premises. The application can illustrate how different users can communicate through a network that is not connected to the internet.

Summary

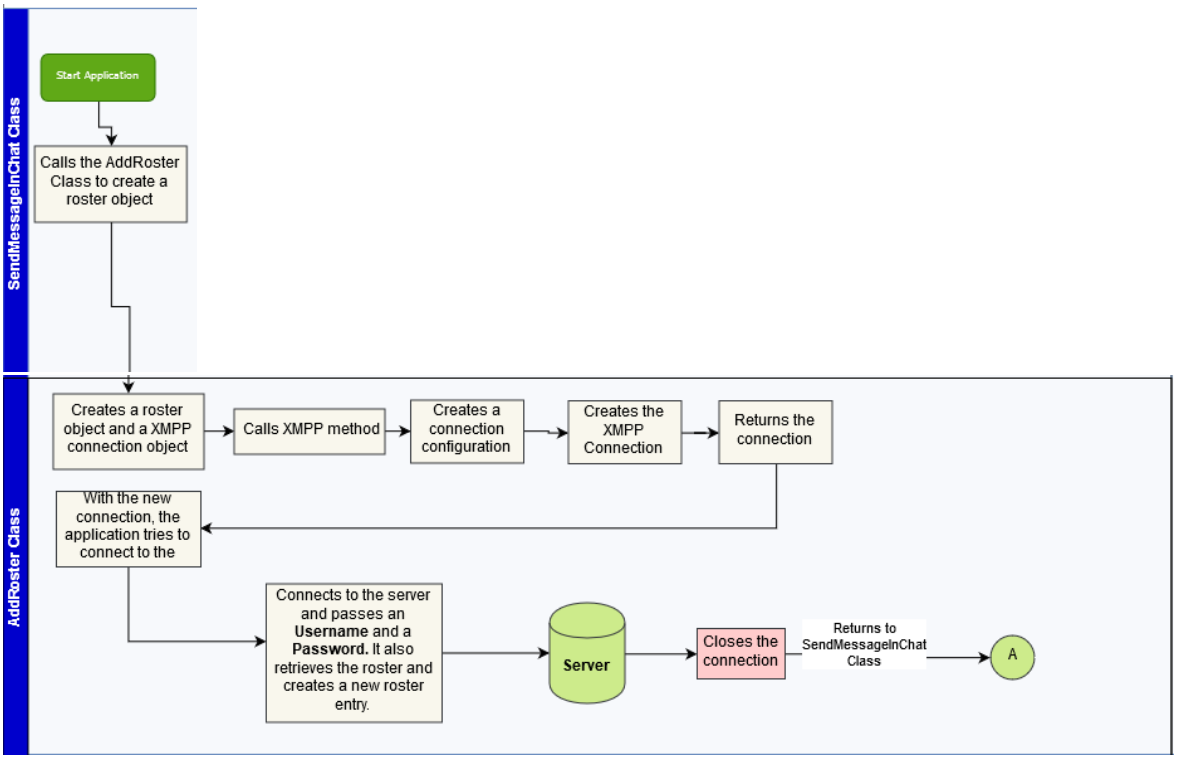
The application creates a connection between two devices over a network that need to communicate with each other. The connection is first stablished before any data can be transmitted. Before stablishing the connection, the application needs to log into the server. To log into the server, the application uses the host id and a port number. Once it connects to the server, it checks for a specified username and a password. The server looks through its roster (contact list) to make sure the username and password being used to log in exist. If it does exist, the application logs into the server. The application then creates a chat specifying who the receiver is by username. Once the chat and the connection to the valid username is stablished, the sending and receiving of messages can begin.

Expectations and Assumptions

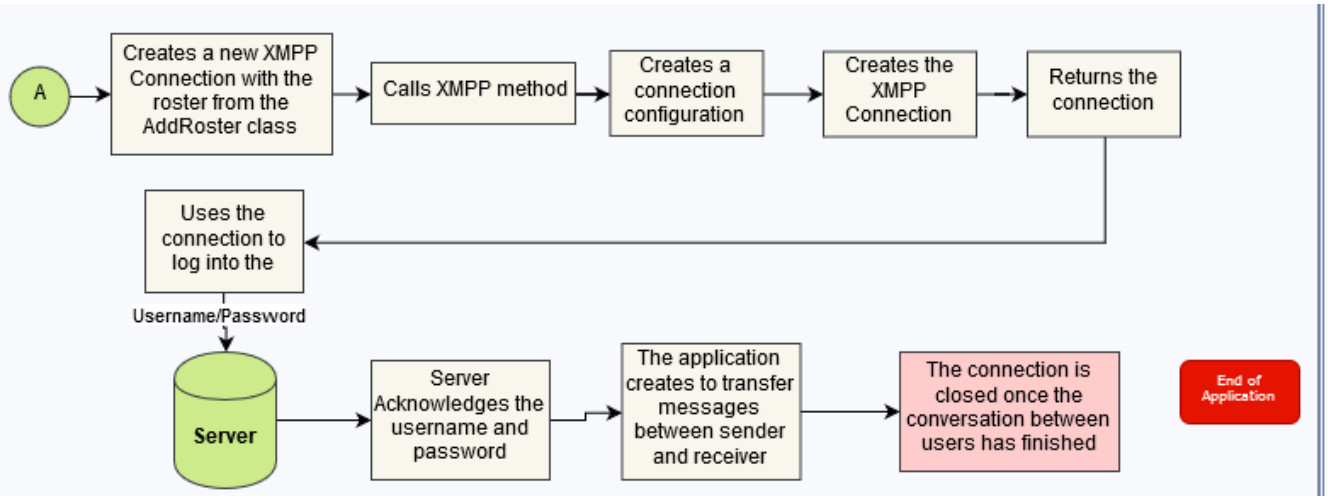
- The application is expected to be able to send and receive text messages using two devices (computers) that are connected to the same network.
- The server is assumed to be already set up and functioning properly and be populated with valid username and passwords.
- Firewalls has been configured properly, if needed, to allow communications between the two devices on the network.
- There is a known receiver and sender

Process

- The java file named SendMessageInChat.java is run to start the application process

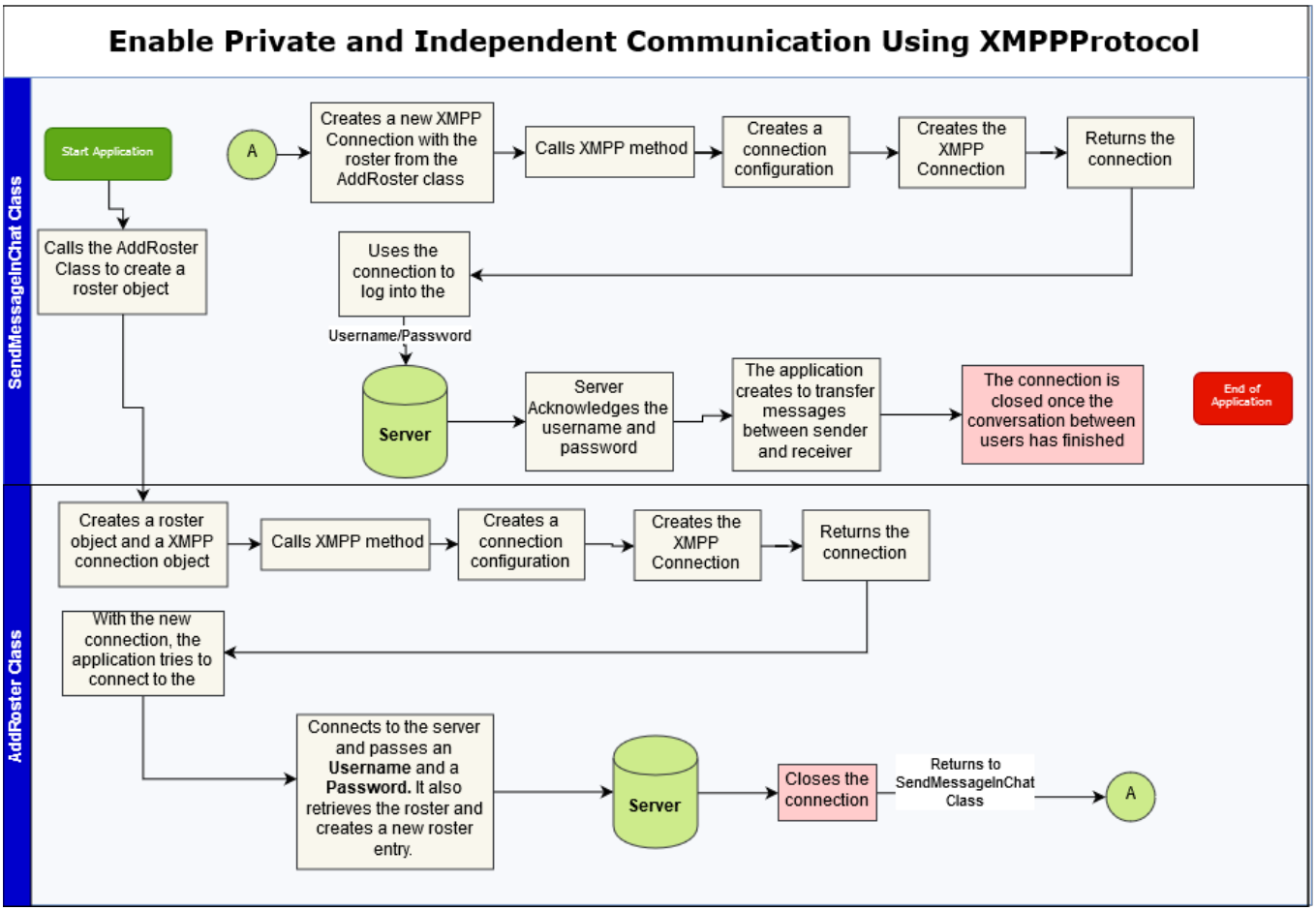


- The application starts by adding a roster to the server.
    - The application establishes a connection with the server using a host id and a port number
    - The application tries to log in using a username and a password
    - It gets the current roster list (contact list)
    - It then creates an entry for the username accessing the server
        - The creation of the entry into the server will allow the user to log in and chat with another user on the server
    - Once the entry in the roster has been created, the connection is closed



- Using the roster previously created, the user creates another connection to the server.
    - Utilizing the new connection, the user logs in using the credentials stored in the server (username and password)
        - The server checks if the username and the password entered exists and matches the information on the server
- **A chat is created**
    - The chat enables the incoming and outgoing communication to exist
    - The chat is created using a UserJIID (username) and an event listener.
        - The userJIID is the destination user
        - The destination user must be a valid user and it is formatted as follows: [Username]@[HostID]
            - Username represents the name of the username as it appears on the server
            - HostID represents the IP address of the server on the network
        - The event listener (message listener) processes incoming messages received
            - The application retrieves the information from the XMPP stanza and stores it in its appropriate values to be able to display it to the user.
- The application keeps a connection open with the selected destination user and ask the user to send the message it wants, while listening for incoming messages
- The application displays the messages

Process Diagram



Objects

Class: AddRoster.java

1. The necessary imports are brought in from the Smack Library:
    a. ConnectionConfiguration: To set up the connection
    b. Roster: The list of addresses
    c. XMPPConnection: Allows the connection to the XMPP server
    d. XMPPException: catches any errors return from the XMPPConnection
2. **Main()**
    a. Creates a Roaster object "**addRoster**", which will store the address of the user on the server.
    b. Creates an XMPPConnection object "**connection**", which will take the **addRoster** variable to connect to the XMPP server
    c. **connection** tries to log into the server using the XMPP connection
        i. Passes an username to the server
        ii. Passes a password to the server
        Note: The server will check the username and password matches an username and a password on the server.
    d. Variable **roster** is declared as an object of Roster
        i. **roster** will store the usernames on the server.
        ii. **roster** will use the **connection** variable to retrieve the list of usernames
    e. **roster** will then create an entry for the user that is trying to communicate to another user on the server
        i. **roster** will use an username to create the Roster entry
        ii. **roster** will use a name for the username where the Roster entry is being created
        iii. **roster** will assigned this username to a group
            1. if no group is selected, the assigned group will be **null**
        iv. once the **roster** entry has been created, the connection is closed
            1. **connection** variable is used to disconnect.
3. XMPPConnection Connect() method
    a. Creates a ConnectionConfiguration object named **config.**
        i. **config** stores the IP address of the server (host) and the application address (port number)
    b. Creates XMPPConnection object named **connection.**
        i. **connection** uses the **config** variable and its IP address and port number to connect to the server.
    c. **connection** tries to connect to the server

Class: SendMessageInChat.java

1. The necessary imports are brought in from the Smack Library:
    a. Chat: Used to create the chat object
    b. ConnectionConfiguration: Used to set up the connection
    c. MessageListener: Used to listen for incoming messages

d. XMPPConnection: Allows the connection to the XMPP server
e. XMPPException: catches any errors return from the XMPPConnection
f. Message: XMPP stanza containing the data that being transfer between sender and receiver.

2. Main()

a. Creates a Roaster object "**addRoster**", which will store the address of the user on the server.
b. Creates an XMPPConnection object "**connection**", which will take the **addRoster** variable to connect to the XMPP server
c. **connection** tries to log into the server using the XMPP connection
    i. Passes an username to the server
    ii. Passes a password to the server
    Note: The server will check the username and password matches an username and a password on the server.
d. Chat object, **chat**, is created
    i. **chat** is to create a new chat
        1. requires an username in the format [**username**]@[**IP Address**]
            a. **username**: name of the actual user as it appears on the server
            b. **IP Address:** requires server destination IP
                i. Accepts IPV4 and IPV6 formats
        2. Creates a new **MessageListener**() anonymous method
            a. **processMessage**() anonymous subclass method which takes an object chat and a message (stanza)
                i. Declares variable **from** which stores the sender
                ii. Declares variable **body** which stores the message
                iii. Checks if **body** is not empty and prints the message from the sender if not empty
        3. Creates a Scanner object, **input**
        4. Loops through **input** while is not empty
            a. **msg** variable stores the user's input
            b. The message is then sent to the recipient using **chat.sendMessage(msg)**
        5. While loop to keep the application running
            a. While loop runs forever so that the recipient and destination can exchange messages
            b. **Thread.sleep** pauses the program so that it does not utilizes all of the device's and network resources
        6. Finally the connection ends

3. **XMPPConnection Connect()** method

a. Creates a ConnectionConfiguration object named **config.**
    i. **config** stores the IP address of the server (host) and the application address (port number)

b. Creates XMPPConnection object named **connection.**
    i. **connection** uses the **config** variable and its IP address and port number to connect to the server.
c. **connection** tries to connect to the server