GROUP PROJECT 2

Group : G4

Leader :

- E. W. I. C. Siriwardana 138235P

Group Members:

- K. L. Dasun 138205B
- H. M. Jayasiri 138215F
- R. S. Ranwala 138227T
- E. W. I. C. Siriwardana 138235P

(A)

A) Problem: Anybody with a client program can connect to a server and start sending messages impersonating somebody else, since there is no password protection.

To overcome this, we decided introduce passwords. The users have to create an account in the first time and they can use it when they log into the system later. Without the password the user id will not be able to be impersonated.

B) Background information: The features described in the document are to be implemented on "Simple Chat" program.

Issues considered

Issue 1: Can someone use a user id he wishes:
Decision: Anyone can use any user id provided, if it is not already taken by someone else.

Issue 2: We want the system to avoid a user being impersonated by some other user.

Option2.1: Let users have user names and passwords.
Disadvantage: Most of the users that come to public chat will find this annoying and time consuming to create a user account.
Option2.2: Let user ids be unique at any given time.
Advantage: This can avoid impersonating inside a session with a simple implementation. Users can instantly connect and start chatting.

Decision: Select option 2.1

Issue3: How does a user get to know whether a user id is available to use?
Decision: When a user run the command to create a user account, he will receive an error message if it already exists. Then he will have to retry with another userid.

Issue4: How does a user creates an account and login to the system?
Decision: A user can create an account using a name available using the following command. **#createaccount <username> <password>.** If the operation is successful, he will be notified and then he can login using **#login <username> <password>.** If login is successful, he can continue chatting or else retry to login.

Issue5: What happens if a user has lost his password?
Decision: He will have to contact a system admin and get his password reset. The default password will be **s1mpl3.ch@t**

Issue6: How does a user change his password?
Decision: He can change password by using the following command. **#changepassword <oldpassword> <newpassword>.** It will be in effect from the next login.

C) **Environment and System Models:** The feature will be implemented as a feature on phase 2 of the chat client program.

D) **Functional Requirements**

- Users are allowed to manage user names and passwords.
- Users may use any user id which is not already in use.
- User can create an account using the **#createaccount <username> <password>** command.
- If the above command is successful, user can login to the system using the **#login <username> <password>** command.
- If the user name already exists, an error message is given.
- User may contact the system admin to reset the password.
- User may change the password using the **#changepassword <oldpassword> <newpassword>** command and It will be in effect from the next login.

(B)

A) **Problem:** There is no facility to send a private message to a particular user. It is useful to have such a facility.

We have decided to give this facility by letting the user to specify whether it is a private message or not in the message itself.

B) **Background information:** The features described in the document are to be implemented on "Simple Chat" program.

Issues considered

Issue1: User should be able to separately send private and public messages.
Decision: User can specify that a message is private by sending the message in the following format**. #private <recipientid> <message>**

Issue2: Does the recipient see any difference between public and private messages he gets?

Option2.1: No. all the received messages will be seen in the same format. He will not see a difference.

Option2.2: Yes. There is a difference. Since he needs to know whether it is a private message when answering, he needs a way to distinguish between public and private messages.

Decision: Option 2.2. Message will contain a field indicating it was sent only to him.

C) **Environment and System Models:** The feature will be implemented as a feature on phase 2 of the chat client program.

E) Functional Requirements

- User is allowed to send private messages.
- By using the **#private <recipientid> <message>** command user can explicitly state that this message should be a private one.
- Any private message is sent to the recipient with the **<sender> private: <message>** format so that the recipient can easily differentiate a private message from a public one.
- User may send ordinary public messages to all recipient by merely typing the message in the console and pressing the enter key.

(C)

A) Problem: It would be nice if instead of all connected users participating in the same global chat session, separate channels could be established. All messages in a channel would be broadcasted to all other clients in that channel, but not to clients outside the channel

To facilitate this, we have decided to introduce channels. A message sent to a channel will be seen by only the members of the same channel. Anyone can join to any channel.

B) Background information: The features described in the document are to be implemented on "Simple Chat" program.

Issues considered

Issue1: Can a user join many channels in the same time?

Option1.1: Yes. Users can join many channels in the same time as they wish.

Option1.2: Disable this possibility because it can make someone read from one channel and sending it to the other which is a privacy issue for channel concept. The users need privacy even between channels

Decision: Option 1.1. Because, even if we allow to join only one channel at a time if he wants he can join one channel, communicate, get out of it and join the other and pass that message. This simply makes the system not user friendly. So users can join many channels as they wish.

Issue2: Users need a simple way to join a channel.
Decision: User can join a channel simply by sending #channel <channelId>. If there is a channel in the given name he will be joined to it. If there is no such channel, one will be created and he will be joined to that. In this case he will receive a message indicating that a channel was created and he was added to that.

Issue3: Members of a channel need to know when somebody joins the channel. Otherwise they will not know who will see the messages.
Decision: When somebody joins a channel all the members of it will receive a message saying that someone with this ID joined

Issue4: Sometimes users may need to know all the members in the middle. Not only at the time they joined.
Decision: They can get the whole list of members of a channel by the following command. **#channelmembers <channelId>**

Issue5: users should be able to see a list of available channels to connect.
Decision: Use **#channellist** to get a list of available channels

Issue6: After some time user might need to leave a channel or channels.
Decision: User can leave channels by repetitively using the following command **#leavechannel <channelid>**

Issue7: User also needs to know what channels he is currently a member of.
Decision: use **#channelsiamin** to get a list of channels that he is a member of

C) **Environment and System Models:** The feature will be implemented as a feature on phase 2 of the chat client program.

D) **Functional Requirements**

- Users are allowed to manage and maintain channels. A user can join any number of channels simultaneously as he wish.
- A user can merely join a channel by typing the **#channel <channelId>** command on the console.
- If a channel with the given <channelid> already exists user is added to it. Else a new channel with the given <channelid> is created and user is added intern.
- When someone connects to a channel all the members are notified using the following message. <userid> joined to the <channelid>
- Users can get a list of all members of a channel by the following command at any time. **#channelmembers <channelId>**
- Use **#channellist** to get a list of available channels.
- A user may leave a channel by using **#leavechannel <channelid>** command.
- **#channelsiamin** command gives a list of channels that current user is a member of.

(D)

A) **Problem:** A user wants to have somebody else monitor her incoming messages while he is in a meeting.

To provide this facility we are introducing a message forwarding feature. A user can forward the messages he is receiving to someone else he wishes.

B) **Background information:** The features described in the document are to be implemented on “Simple Chat” program.

Issues considered

Issue1: Does the other party aware of this? That someone is going to forward him messages.
Decision: He will receive a notification mentioning that he has been selected by the actual receiver to forward his messages to.

Issue2: Does the original recipient receives the message or only the one to whom it is forwarded gets it?
Decision: Both of them will receive it.

Issue3: How the user selects to whom to forward messages?
Decision: User can activate forwarding by the following command. **#forward <userid>**

Issue4: Is there a way that he can know that his messages are forwarded
Decision: The message will have a field indicating it is being forwarded to the given user.

Issue5: Can he cancel forwarding
Decision: Yes he can, by using the following command **#cancelforward <userid>**

Issue6: Can the person who receive the forwarding message reply for those messages
Decision: No. He can only receive. This is because the original receiver is forwarding because he is not available and he will not be aware of the impersonated replies.

Issue7: Can somebody forward messages to some id which is not in use?
Decision: No. He will simply receive an error message indicating that there is no such user.

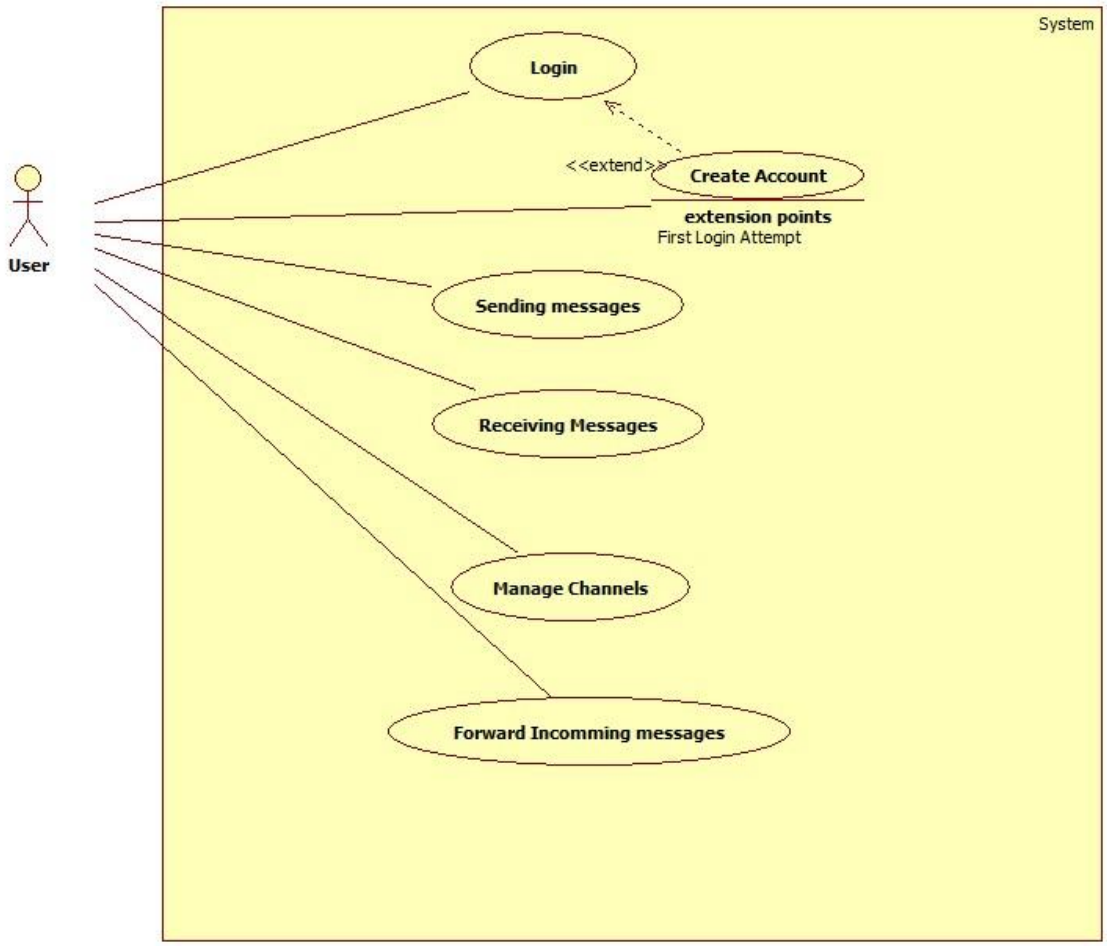
Issue8: Can somebody forward messages to another who is offline?
Decision: He will receive an error message indicating that the user is offline.

C) Environment and System Models: The feature will be implemented as the forth feature on phase 2 of the chat client program.

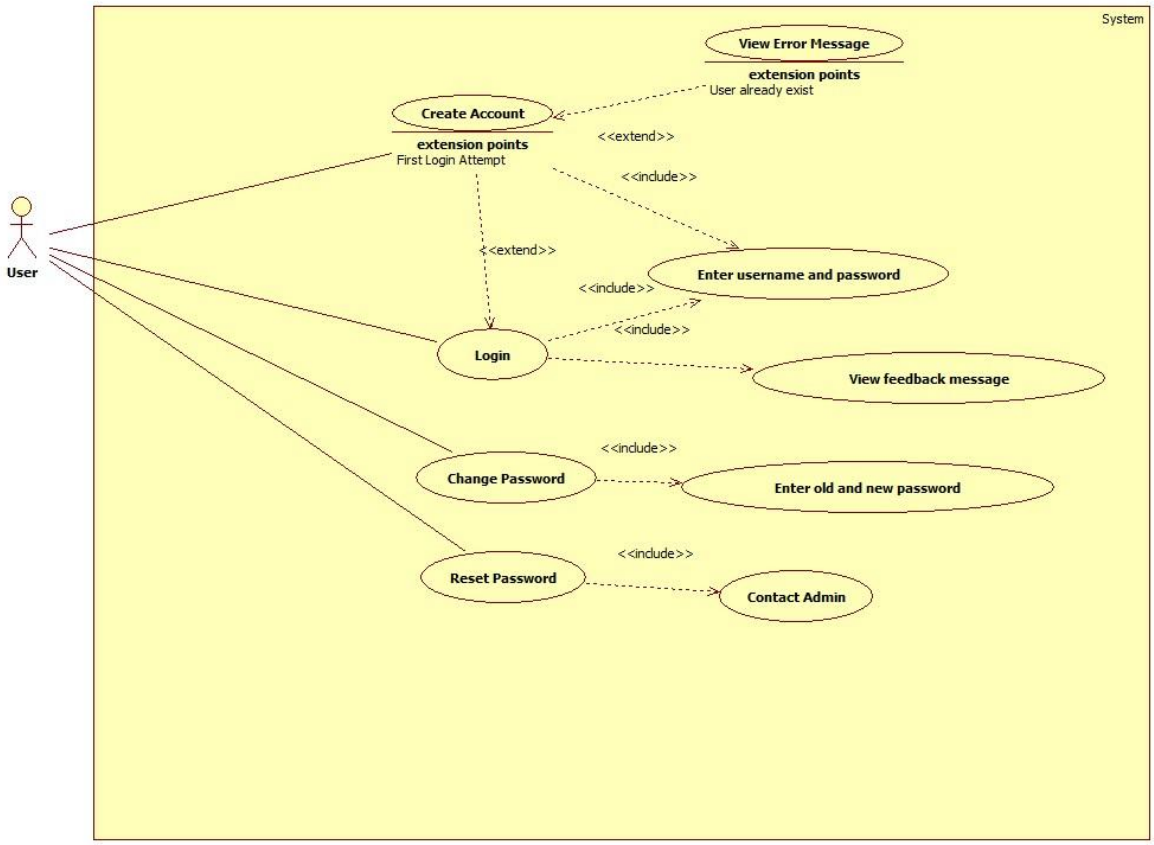
D) Functional Requirements

- User is allowed to forward the messages he received to someone else he wishes since he is in a meeting.
- User can activate forwarding by typing **#forward <userid>** command
- Other party needs to be informed regarding this. So the following notification message is sent. **You have been chosen by <userid> to forward his messages.**
- Both of the original recipient and the other one receives the message.
- Original user can cancel forwarding at any time by issuing **#cancelforward <userid>**command.
- The other user is notified regarding this using the **<userid> has cancelled the forwarding of her messages to you**.
- The second user cannot reply and forward these messages to anybody else in the meantime.

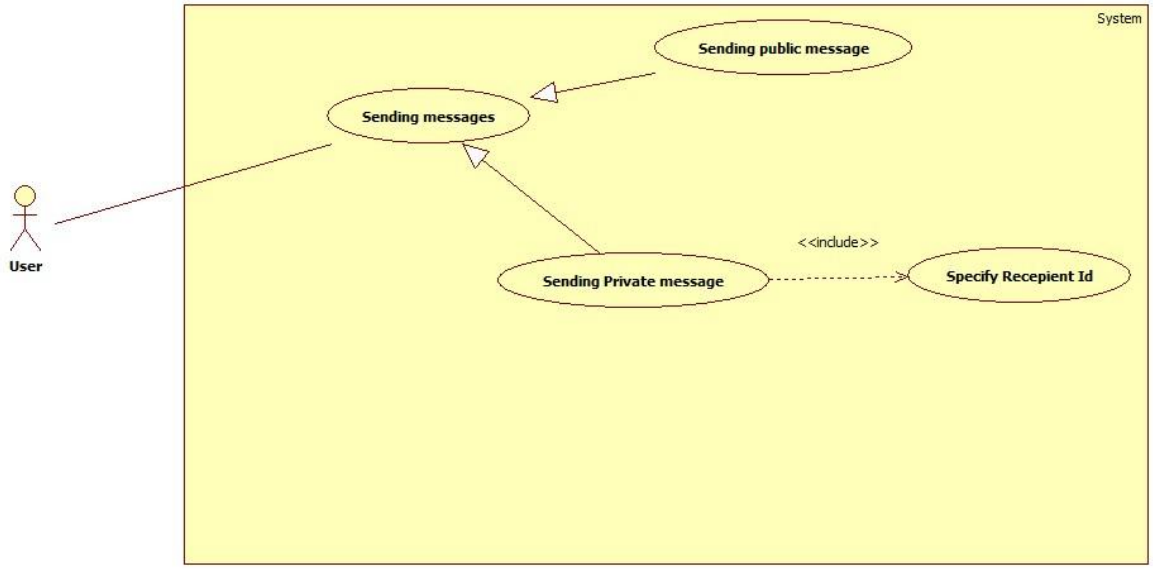
High Level Use case Diagram



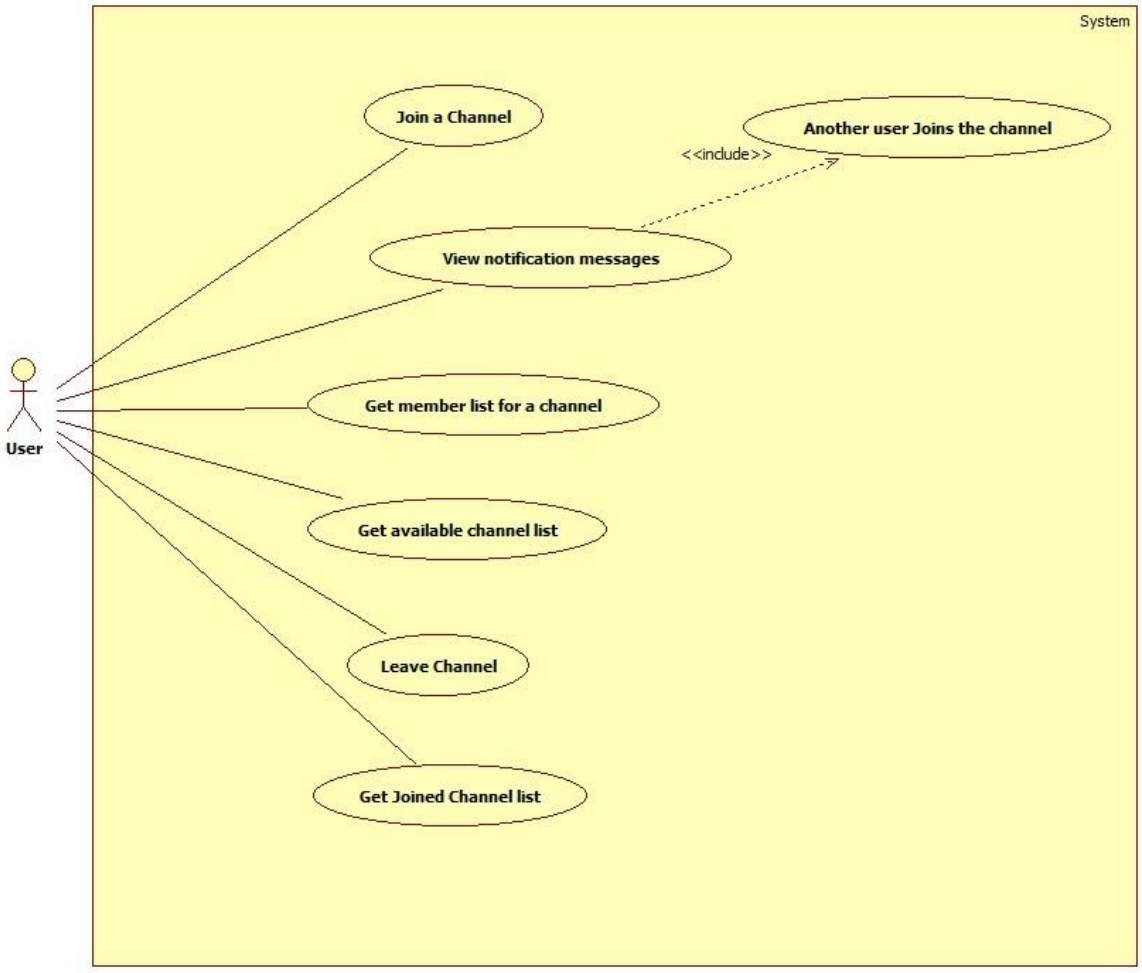
Detailed Use case Diagram for the Login Scenario



Detailed Use case Diagram for the Sending messages scenario



Detailed Use case Diagram for the Maintain Channels scenario



Detailed Use case Diagram for forwarding messages scenario