Group-to-MUC implementation.

The Group-to-MUC provides:

1. Simplified chat group creation.
2. The roster based on the group membership.
3. Presence notifications based on the group membership.
4. Group timeline.
5. Filtering of group timeline events.

Note: throughout the text and the code that follows, we will use:

'userid' – for the user id used by Showdme;
'groupid' or 'groupid-1', 'groupid-2', ... – for the group id(s) used by Showdme;
'resourceid' – for the session resource used by the current web client session;
'iq-id' – for the randomly generated id used in <iq> stanzas;
'memberid-1', 'memberid-2', 'memberid-3', ... - for the member ids of the Showdme group.

1. Simplified chat group creation.

To create (or join) the group, the authenticated client sends presence stanza:

```
<presence to="groupid@conference.showdme.com/userid" >
        <x xmlns="http://jabber.org/protocol/muc"/>
</presence>
```

In case of success, the ejabberd responds with presence stanza containing the group membership information:

```
<presence from="groupid@conference.showdme.com/userid" xml:lang="en"
to="userid@showdme.com/resourceid" >
<x xmlns="http://jabber.org/protocol/muc#user">
<item affiliation="owner" role="moderator" jid="userid@showdme.com/resourceid" />
<status code="110" />
</x>
</presence>
```

The permanent group will be created, if doesn't yet exist, and the user will be given permanent membership in there. If the group has been previously created and/or the user has been previously given a membership, sending presence will not have any effect.

Note: The creation of the group from web client replaces the same operation that is currently

being performed by xmppAdmin account on Node.js side.

Note: Contrary to the standard MUC implementation, the client will not receive group-based presence notifications from other group members. Instead, the presence notifications will be routed through group-based roster (see p.2).

Directions for the web client implementation:

To create/join the group, send the presence stanza to the group id, as described above. Provide the appropriate Strophe callback handler code to ensure that the group was successfully created.

Please note, that the XMPP exchange will not use human-readable information about the group and the user, such as the group title and the user name, instead group id and/or user id will be used. Please use appropriate web backend API calls to retrieve such information by group/user id.

2.The roster based on the group membership.

The roster will contain the list of users who share the same groups with the current session user. "Sharing" applies to the membership, and not the current presence in the group. In other words, the users who are showing in the roster have a permanent membership in the group, which was created by them explicitly joining the group (see p.1).

The roster is available through the standard XMPP request, and the roster list organized by the group id.

To obtain the roster, the authenticated user sends:

```
<iq type="get" id="iq-id" >
        <query xmlns="jabber:iq:roster"/>
</iq>
```

The ejabberd responds with:

```
<iq from="userid@showdme.com" type="result"
to="userid@showdme.com/resourceid" id="iq-id" >
<query xmlns="jabber:iq:roster">
<item subscription="both" name="userid" jid="userid@showdme.com" >
<group>groupid-1</group>
</item>
<item subscription="both" name="memberid-2" jid="memberid-2@showdme.com" >
<group>groupid-1</group>
</item>
<item subscription="both" name="memberid-3" jid="memberid-3@showdme.com" >
<group>groupid-1</group>
```

```
</item>
<item subscription="both" name="userid" jid="userid@showdme.com" >
<group>groupid-2</group>
</item>
<item subscription="both" name="memberid-3" jid="memberid-2@showdme.com" >
<group>groupid-2</group>
</item>
<item subscription="both" name="memberid-4" jid="memberid-2@showdme.com" >
<group>groupid-2</group>
</item>
<item subscription="both" name="userid" jid="userid@showdme.com" >
<group>groupid-3</group>
</item>
</query>
</iq>
```

In the example above, the roster consists of the members of 3 groups ('groupid-1', 'groupid-2' and 'groupid-3'). The 'userid' itself (i.e., the user that has requested the roster) is present in all 3 groups, at it obviously has to be a member thereof. 'userid' is currently the only member of 'groupid-3'. Also, 'memberid-3' shares 'groupid-1' and 'groupid-2' with the current user, whereas 'memberid-2' and 'memberid-4' share 'groupid-1' and 'groupid-2', respectively.

Directions for the web client implementation:

To retrieve the roster, send the <iq> stanza as described above. Provide Strophe iq handler, based either on 'iq-id' or the 'jabber:iq:roster' namespace in order to handle the above described response from ejabberd. Use web backend API to obtain user and group names for the UI.

Note: the order of records in the roster output is not currently guaranteed to be sorted by group id.

Note: it is advisable to retrieve the roster before the initial presence is sent from the session, as this will allow to match incoming presence notifications agains the roster list (see below for more information).

3. Presence notifications based on the group membership.

Once the authenticated user session advertises the presence (by sending 'available' presence stanza to ejabberd), it will start to automatically receive presences from the user sessions that share the same groups. This applies both to the users who had joined the shared groups before the current session started ('old' users), and to the ones who joined shared groups during the time the current session was active ('new' users). However, the presence handling for these two groups of users will be slightly different.

- for the 'old' users, the 'available' presence notifications will be sent to the as follows:

<presence from="memberid-1@showdme.com/memberid-resorceid"
to="userid@showdme.com/resourceid" ></presence>

- for the 'new' users, the “roster push” stanza will be received by current session, followed by the 'available' presence, as follows:

<iq from="userid@showdme.com" type="set" to="userid@showdme.com/zephyr" id="iq-id" >
<query xmlns="jabber:iq:roster">
<item subscription="both" name="new-memberid" jid="new-memberid@showdme.com" >
<group>groupid1</group>
</item>
</query>
</iq>

<presence from="new-memberid@showdme.com/new-memberid-resorceid"
to="userid@showdme.com/resourceid" ></presence>

The above example notifies the session that the 'new-memberid' user has just joined the group 'groupid1'.

When the member of any shared group goes offline, the user session receives 'unavailable' presence:

<presence from="memberid-1@showdme.com/memberid-resorceid" type=”unavailable”
to="userid@showdme.com/resourceid" ></presence>

When the member of any shared group leaves the group, the user session receives 'roster push' notification as follows:

<iq from="userid@showdme.com" type="set" to="userid@showdme.com/zephyr"
id="iq-id" >
<query xmlns="jabber:iq:roster">
<item subscription="none" name="new-memberid" jid="new-memberid@showdme.com" >
<group>groupid1</group>
</item>
</query>
</iq>

Directions for the web client implementation:

To handle online presence, create or modify Strophe callback that handles presence stanzas according to the needs of UI.

To handle roster push notifications, create or modify the Strophe IQ callback for “jabber:iq:roster” namespace.

4. Group timeline.

Group timeline will contain the items that have been broadcasted to the group. The system will not impose the content of the items, as long as it conforms to the special <message> stanza format as described below.

To send the item to the group, the authenticated user sends:

```
<message type="groupchat" to="groupid@conference.showdme.com" id="a125ba"
content="content-type">
<body>content-specific-data</body>
</message>
```

The ejabberd will broadcast the item to all online members, including the sender:

```
<message from="groupid@conference.showdme.com/userid" type="groupchat" xml:lang="en"
to="memberid@showdme.com" id="a125ba" content="content-type">
<body>content-specific-data</body>
</message>
```

The ejabberd will handle the items transparently, meaning it will broadcast all the messages to the online group members as is. The <message> stanzas will be used as a carrier of the group items. The attribute 'content' will be used for filtering of the items if the user has requested the filtering (see p.5 for details).

The <body> can contain any type of data, provided it adheres to XML rules (i.e. properly escaped or wrapped in CDATA). The web client code is responsible for creating of the outgoing <message> stanzas and parsing and interpretation of the data enclosed in the <body> element of incoming <message> stanzas, by using the value of 'content' attribute.

> Directions for the web client implementation:
>
> Create the Strophe handler for the incoming messages in the format described above, implement parsing of <body> element according to the value of 'content' attribute. Note: only the messages that have type='groupchat' should be handles, any other messages coming from the JID that contains @conference.showdme.com have to be ignored.

5. Filtering of group timeline events.

By default, all the broadcasted items will be delivered to online group members. In order to subscribe for the certain groups' content and events, the authenticated client sends to the group:

```
<iq to="conference.showdme.com" type="set" id="iq-id">
```

```
<query xmlns="showdme:group:subscribe" >
        <groups>
                <group groupid='groupid-1' content-types='content-types'
                        event-types='event-types' />
                <group groupid='groupid-1' content-types='content-types'
                        event-types='event-types' />
        </groups>
  </query>
</iq>
```

The attributes 'content-types' and 'event-types' will contain comma-delimited content and event types that the client wish ejabberd to subscribe to for each group within <groups/> element.

This will register session-based subscriptions for the session which has sent the request. The ejabberd will respond to acknowledge:

```
<iq from="conference.showdme.com" type="result">
  <query xmlns="showdme:group:subscribe">
        <success/>
  </query>
</iq>
```

Note: there will be no notifications sent to the client session from any of the groups it joined but didn't sent the subscription request to.

Note: each such consequent request will overwrite the subscriptions. To cancel all subscriptions, the client session will send the 'iq' request with empty <groups> content, like so:

```
<iq to="conference.showdme.com" type="set" id="iq-id">
  <query xmlns="showdme:group:subscribe" />
        <groups />
  </query>
</iq>
```

Note: ejabberd will not verify the content types, the web application will be responsible for forming content type lists that correspond to actual content type being used.

Directions for the web client implementation:

Optionally implement Strophe handler for 'showdme:group:subscribe' namespace to make sure the filter was accepted by ejabberd.