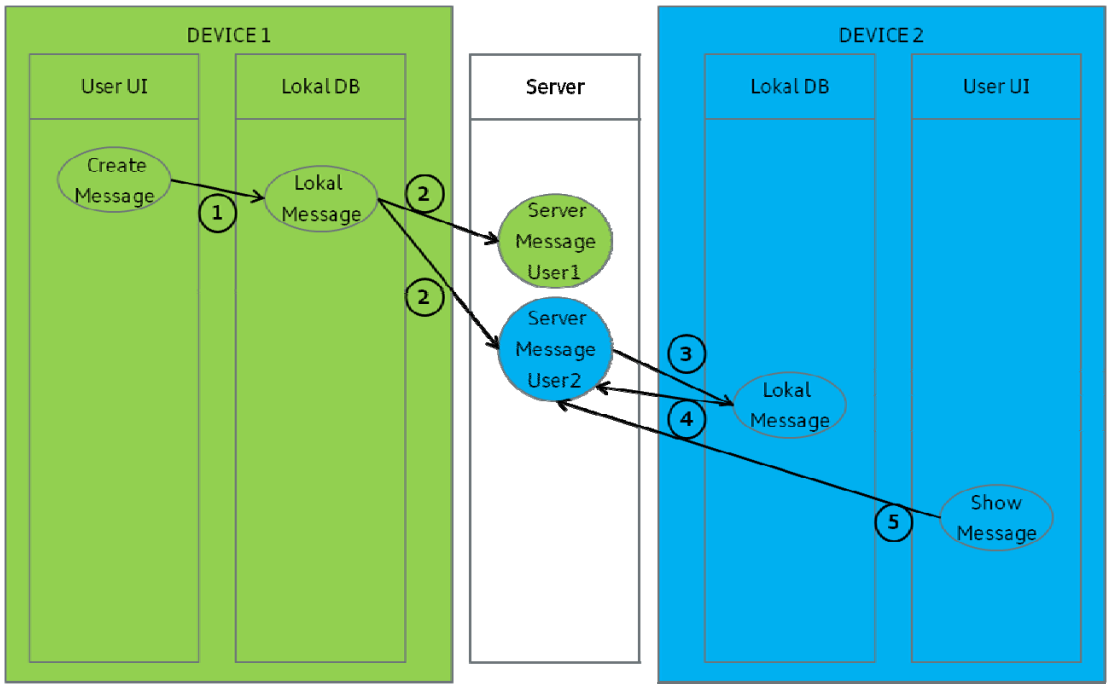
Kommunikationsbeziehungen

1. 

Zuerst wird die Nachricht lokal auf dem Android Gerät gespeichert. In der lokalen SQLite Datenbank. Dort wird zwar für die Listviews eine _id vergeben, aber die BackendID welche von der Server Datenbank kommt wird mit 0 belegt.

SingleChatActivity.java

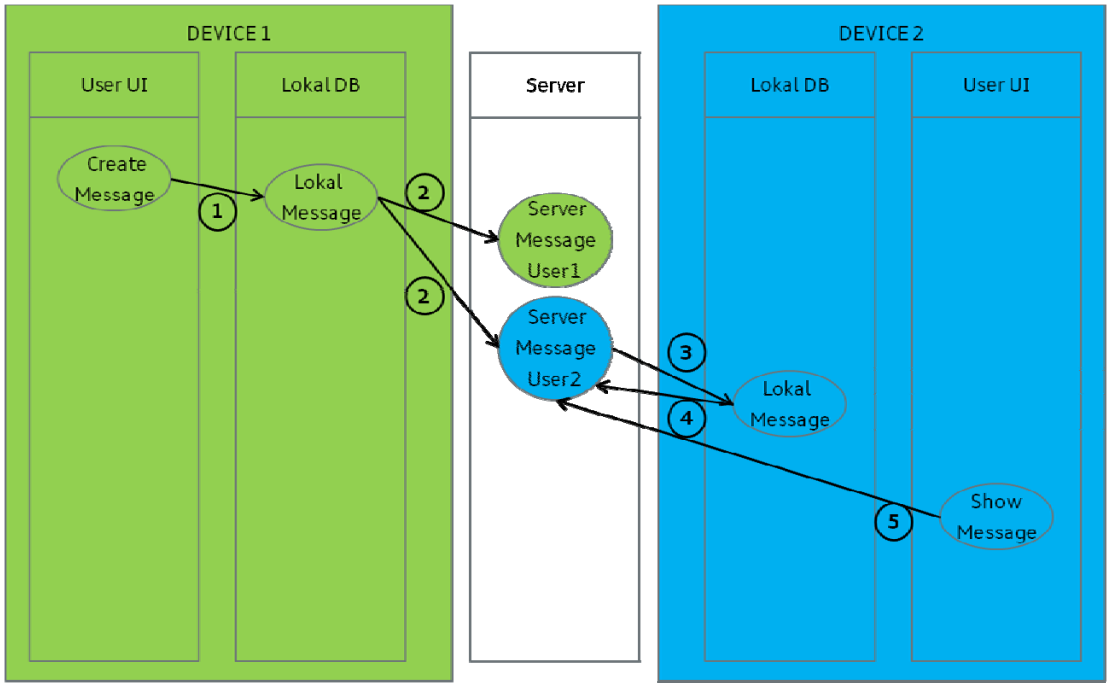
```
Message = (EditText) findViewById(R.id.chatText);
        Button send = (Button)
findViewById(R.id.buttonSend);
        send.setOnClickListener(new
View.OnClickListener() {
            public void onClick(View arg0) {
                if (Message.getText().toString().length()
> 0) {

                    Intent iSendTextMsgService = new
Intent(SingleChatActivity.this, MeBaService.class);


iSendTextMsgService.setAction(Constants.ACTION_SENDTEXTME
SSAGE);

iSendTextMsgService.putExtra(Constants.CHATNAME,
ChatName);

iSendTextMsgService.putExtra(Constants.CHATID, ChatID);

iSendTextMsgService.putExtra(Constants.USERID, userid);
                    String tmpmsg =
Message.getText().toString();
```

```
iSendTextMsgService.putExtra(Constants.TEXTMESSAGE,
tmpmsg);

                    startService(iSendTextMsgService);

                    Message.setText("");
                    Message.clearFocus();
                }
            }
        });
```

MeBaService.java

```
private void insertNewMsgIntoDB(int ChatID, int UserID,
String MessageType, String Message) {
        Log.d(TAG, "start insertMsgIntoDB");

        // Insert new Message into local DB and trigger
Sync to upload the Information.
        // To find the not send messages the Backend ID
musst be 0 and the
        // Sendtimestamp musst be 0
        // The Readtimestamp and the MessageIDs are
supplied by the Server
        // The ChatID is needed to insert the Message
into the right Chat afterwards

        long time = System.currentTimeMillis() / 1000L;

        ContentValues valuesins = new ContentValues();
        valuesins.put(Constants.T_MESSAGES_BADBID, 0);
        valuesins.put(Constants.T_MESSAGES_OwningUserID,
UserID);

valuesins.put(Constants.T_MESSAGES_OwningUserName,
username);
        valuesins.put(Constants.T_MESSAGES_ChatID,
ChatID);
        valuesins.put(Constants.T_MESSAGES_MessageTyp,
MessageType);
        valuesins.put(Constants.T_MESSAGES_SendTimestamp,
time);
        valuesins.put(Constants.T_MESSAGES_ReadTimestamp,
time);
        if
(MessageType.equalsIgnoreCase(Constants.TYP_TEXT)) {
            valuesins.put(Constants.T_MESSAGES_TextMsgID,
0);

valuesins.put(Constants.T_MESSAGES_TextMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_IMAGE)) {
```

```
valuesins.put(Constants.T_MESSAGES_ImageMsgID, 0);

valuesins.put(Constants.T_MESSAGES_ImageMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_LOCATION)) {

valuesins.put(Constants.T_MESSAGES_LocationMsgID, 0);

valuesins.put(Constants.T_MESSAGES_LocationMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_CONTACT)) {

valuesins.put(Constants.T_MESSAGES_ContactMsgID, 0);

valuesins.put(Constants.T_MESSAGES_ContactMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_FILE)) {
            valuesins.put(Constants.T_MESSAGES_FileMsgID,
0);

valuesins.put(Constants.T_MESSAGES_FileMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_VIDEO)) {

valuesins.put(Constants.T_MESSAGES_VideoMsgID, 0);

valuesins.put(Constants.T_MESSAGES_VideoMsgValue,
Message);
        }
        ContentProviderClient client =
getContentResolver().acquireContentProviderClient(Frinmea
nContentProvider.MESSAES_CONTENT_URI);

client.getLocalContentProvider().insert(FrinmeanContentPr
ovider.MESSAES_CONTENT_URI, valuesins);

        Log.d(TAG, "end insertMsgIntoDB");
    }
```

2. 

Diese 0 ist ein Indikator für den SyncService, diese Nachricht ist dem Server noch nicht bekannt und muss versendet werden (uploadUnsavedMessages).

SyncAdapter.java

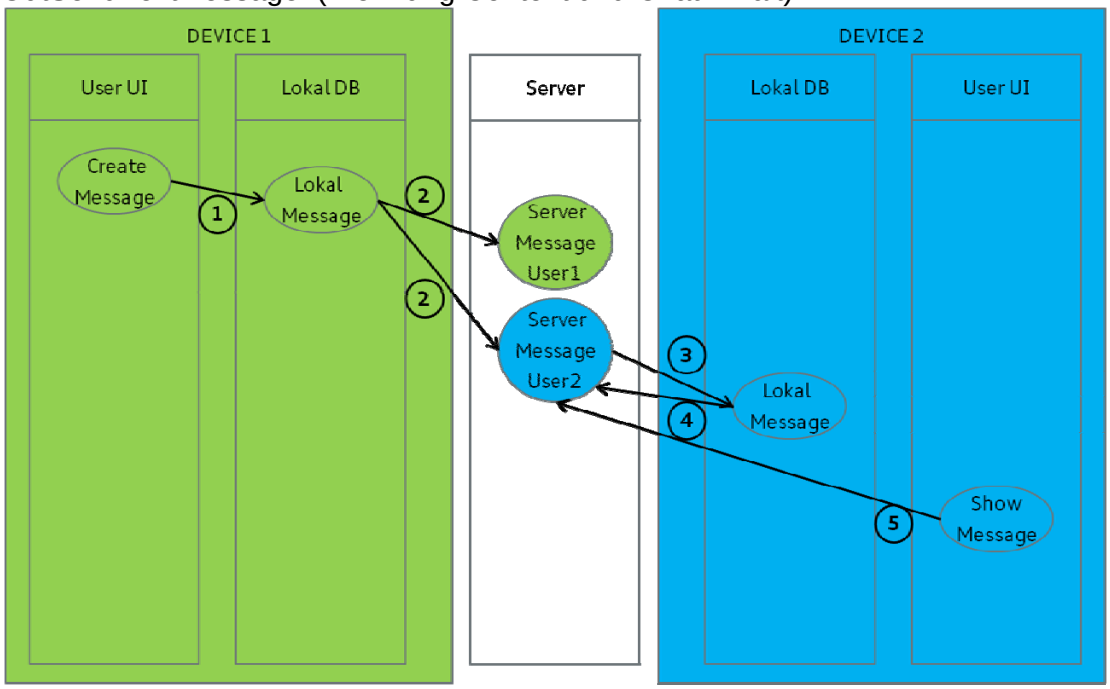
```
private void uploadUnsavedMessages() {

        ContentProviderClient client =
mContentResolver.acquireContentProviderClient(FrinmeanCon
tentProvider.MESSAES_CONTENT_URI);
        Cursor c = ((FrinmeanContentProvider)
client.getLocalContentProvider()).query(FrinmeanContentPr
ovider.MESSAES_CONTENT_URI, MESSAGES_DB_Columns,
T_MESSAGES_BADBID + " = ?", new String[]{"0"}, null);

        while (c.moveToNext()) {

            String msgtype =
c.getString(ID_MESSAGES_MessageType);

            if (msgtype.equalsIgnoreCase(TYP_TEXT)) {
                OutSendTextMessage outtxt =
rf.sendtextmessage(username, password,
c.getString(ID_MESSAGES_TextMsgValue));
                if (outtxt != null) {
                    if (outtxt.getErrortext() == null ||
outtxt.getErrortext().isEmpty()) {
                        OutInsertMessageIntoChat outins =
rf.insertmessageintochat(username, password,
c.getInt(ID_MESSAGES_ChatID), outtxt.getTextID(),
TYP_TEXT);
                        if (outins != null) {
                            if (outins.getErrortext() ==
null || outins.getErrortext().isEmpty()) {
```

```
updateUploadedNessagesDatabase(c.getInt(ID_MESSAGES__id),
outins.getMessageID(), outins.getSendTimestamp(),
outins.getSendTimestamp(), outtxt.getTextID(), TYP_TEXT,
null);
                                }
                            }
                        }
                    }
                }
```

…….

Der Server legt pro User im Chat einen eigenen Eintrag ind er Messages Tabelle an. Deswegen 2 Nachrichten auf dem Server, eine für den Versender, eine für den Empänger.

Der Content wird vor dem Einfügen der Nachricht in den Chat hoch geladen, z.B. für Text mit SendTextMessage und dem Return-Type OutSendTextMessage. (Trennung Content und Chat-Inhalt).

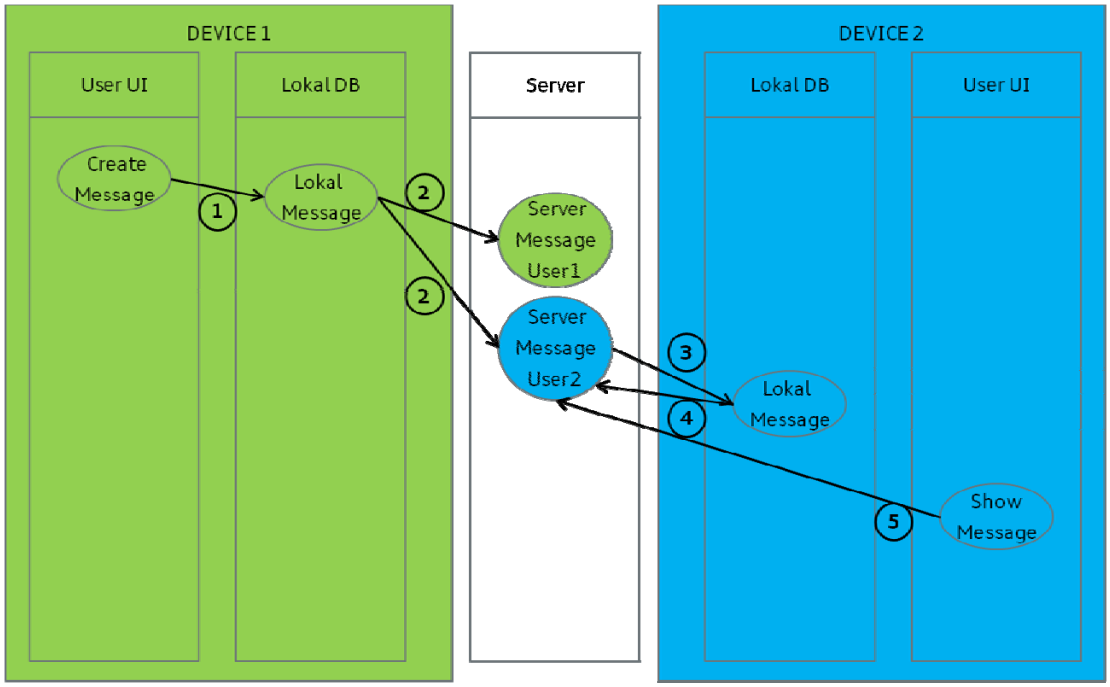
3. 

SyncAdapter checked neue Nachrichten mit (syncCheckNewMessages). Wenn in den Chats neue Nachrichten vorhanden sind, dann werden dies herunter geladen, Message inkl. Content.

Die Funktione SaveMessageToDB übernimmt das durchlaufen aller Nachrichten in einem Chat nud lädt pro Nachrichten-Typ den Content nach, siehe:

```
private void SaveMessageToLDB(List<Message> in, int
ChatID, String ChatName) {
        Log.d(TAG, "start SaveMessageToLDB");
        for (int j = 0; j < in.size(); j++) {
            Message m = in.get(j);
            ContentValues valuesins = new
ContentValues();
            valuesins.put(T_MESSAGES_BADBID,
m.getMessageID());
```

```
            valuesins.put(T_MESSAGES_OwningUserID,
m.getOwningUser().getOwningUserID());
            valuesins.put(T_MESSAGES_OwningUserName,
m.getOwningUser().getOwningUserName());
            valuesins.put(T_MESSAGES_ChatID, ChatID);
            valuesins.put(T_MESSAGES_MessageTyp,
m.getMessageTyp());
            valuesins.put(T_MESSAGES_SendTimestamp,
m.getSendTimestamp());
            valuesins.put(T_MESSAGES_ReadTimestamp,
m.getReadTimestamp());
            if
(m.getMessageTyp().equalsIgnoreCase(TYP_TEXT)) {
                valuesins.put(T_MESSAGES_TextMsgID,
m.getTextMsgID());
                OutFetchTextMessage oftm =
rf.gettextmessage(username, password, m.getTextMsgID());
                if (oftm != null) {
                    if (oftm.getErrortext() == null ||
oftm.getErrortext().isEmpty()) {
                        if
(acknowledgeMessage(Constants.TYP_TEXT,
oftm.getTextMessage(), m.getMessageID())) {

valuesins.put(T_MESSAGES_TextMsgValue,
oftm.getTextMessage());
                            ContentProviderClient client
=
mContentResolver.acquireContentProviderClient(FrinmeanCon
tentProvider.MESSAES_CONTENT_URI);
                            ((FrinmeanContentProvider)
client.getLocalContentProvider()).insertorupdate(Frinmean
ContentProvider.MESSAES_CONTENT_URI, valuesins);
                            client.release();
                        }
                    }
                }
            }
…….
```

4. 

Das Acknowledge der Nachticht setzt erst den ReadTimeStamp auf dem Server, zuvor wurde nur der TempReadTimestamp gesetzt beim ausliefern der Nachricht. Sollte das Acknowledge nicht funktionieren, bliebt die Nachricht auf dem Server in dem Zustand „Ungelesen“ und wird immer wieder mit ausgeliefert. Hier der Aufruf vom Acknowldge nach dem herunterladen der Nachricht für Textnachrichten.

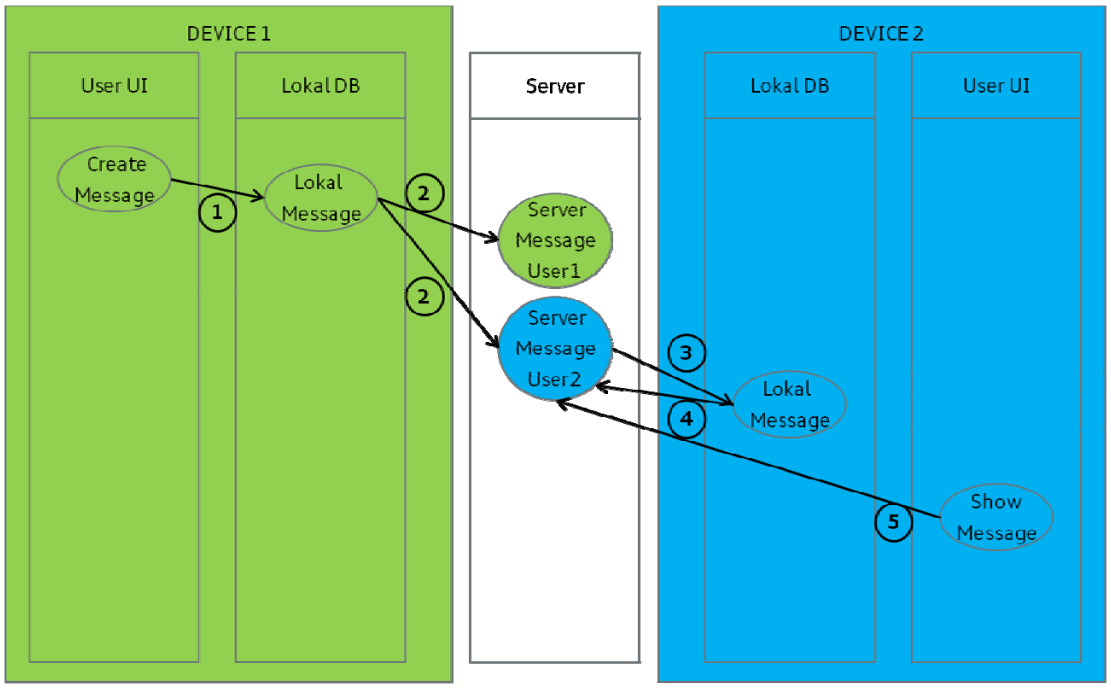
```
……..
if (m.getMessageTyp().equalsIgnoreCase(TYP_TEXT)) {
                valuesins.put(T_MESSAGES_TextMsgID,
m.getTextMsgID());
                OutFetchTextMessage oftm =
rf.gettextmessage(username, password, m.getTextMsgID());
                if (oftm != null) {
                    if (oftm.getErrortext() == null ||
oftm.getErrortext().isEmpty()) {
                        if
(acknowledgeMessage(Constants.TYP_TEXT,
oftm.getTextMessage(), m.getMessageID())) {

valuesins.put(T_MESSAGES_TextMsgValue,
oftm.getTextMessage());
……
```

Hier ein Teil der Acknowledge Funktion mit dem Aufruf des Restful Service

```
private boolean acknowledgeMessage(String msgType, String
message, int msgid) {
        Log.d(TAG, "start acknowledgeMessage");

        boolean ret = false;

        if (msgType.equalsIgnoreCase(Constants.TYP_TEXT))
{
            Hasher hasher = Hashing.sha1().newHasher();
```

```
            hasher.putBytes(message.getBytes());
            byte[] sha1 = hasher.hash().asBytes();

            OutAcknowledgeMessageDownload oack =
rf.acknowledgemessagedownload(username, password, msgid,
sha1.toString());
            if (oack != null) {
                if (oack.getErrortext() == null ||
oack.getErrortext().isEmpty()) {
                    if
(oack.getAcknowledge().equalsIgnoreCase(Constants.ACKNOWL
EDGE_TRUE)) {
                        ret = true;
                    }
                }
            }
        }
```

.....

5. 

Um den Status einer Nachricht feststellen zu können, müssen alle Nachrichten welche durch einen Sendevorgang erstellt werden zusammen betrachtet werden, also hier die Server Message User1 und Server Message User2. Die Nachricht vom Sender hat automatisch einen Send-, Read- und ShowTimestamp. Der SendTimestamp wird im Schritt 2 gesetzt, der ReadTimestamp wird im Schritt 4 gesetzt. Wenn der User jetzt den Chat mit der Nachricht öffnet wird der ShowTimestamp gesetzt.

Jetzt ist die Grundlage für die Funktion getMessageInformation gelegt. Diese Funktioner erhält eine Nachrichten-ID vom Server. Zuerst wird die Gruppe der Nachrichten identifiziert und über die Gruppe werden die Anzahl der Nachrichten mit SendTimestamp != 0, ReadTimestamp != 0 und ShowTimestamp != 0 ermittelt und zurück geliefert.