Lab 3: MQTT-based Twitter App Part I

Design and Development of Mobile Applications
E. De Coninck, S. Leroux, P. Simoens
2015-2016

In this and the next lab session we will build a chat application using the MQTT messaging protocol for mobile-cloud communication. In addition, we will learn to work with `ContentProvider` component.

1 Application Requirements

The Twitter-like application must meet the following requirements:

- Show online/offline state of all app users.
- Subscribe to a user feed.
- Post messages to **your** feed.
- Save received messages to `ContentProvider`. Messages which are send before subscribing can not be received and unsaved messages can not be reacquired from the broker.

In the next session we will provide you with a MQTT broker, but first we will design and implement the front-end application using a provided `ContentProvider`, so you can focus on the development of the mobile app.

2 Twitter-like application

Start from the code provided on Minerva (solution of lab 2 with TODOs for this session). Download source code and **import project** into Android Studio. The `MainActivity` will show all users saved to the ContentProvider, which will later be populated by a MQTT broker. The `DetailActivity` will show the message feed of the selected person. The start code contains an extra java package contentprovider which holds the database structure and the `ContentProvider`.

be.ugent.oomt.labo3.contentprovider Content providers are one of the primary building blocks of Android applications, providing content to applications and sharing data between different applications. They encapsulate data and provide it to applications through the `ContentResolver` interface.

The supplied content provider has a SQLLite database with two tables, `contacts` and `messages`, which can be accessed through the content provider. The columns of each table are as follows:

- Contacts:

  `DatabaseContract.Contact.COLUMN_NAME_CONTACT`: String (id)

  `DatabaseContract.Contact.COLUMN_NAME_STATE`: String

`DatabaseContract.Contact.COLUMN_NAME_LAST_UPDATE`: Date (optional)

- Messages:

  `DatabaseContract.Message.COLUMN_NAME_CONTACT`: String (ref id)

  `DatabaseContract.Message.COLUMN_NAME_MESSAGE`: String

- Add the provided content provider to your app manifest file. This will make sure the SQLLite database is created when your application is first run.
- Update the list view code of last week to take input from the content provider. The `ListAdapter` of `MainFragment` must be replaced with a `SimpleCursorAdapter`. This adapter needs some extra parameters: the cursor which can be null at first but needs to be swapped with the correct cursor, an array of column names from the cursor (accessible through `DatabaseContract.Contact|Message` class and the array of ids to map the columns to text views of the selected layout file. Use the `android.R.layout.simple_list_ item_activated_2` which has 2 text views for the name and the state (`android.R.id.text1|text2`).

  Load the cursor by an asynchronous `LoadManager`. This will make sure your data is always up-to-date and you do not have to re-query if data is added, updated or removed. We will update the `DetailFragment` later with the messages received from an other client (for now, just comment out the show details part).
- To test the content provider insert a test user once with some sample messages into the content provider's database. Use `MessageProvider.CONTACTS_CONTENT_URL` content uri for inserting contacts and `MessageProvider.MESSAGES_CONTENT_URL` for new messages. Add content values for the required fields of each table and keep in mind that messages need a reference to the contact.
- Implement the `DetailFragment` so the title has the contacts name and the second text view contains all messages send to his feed. On selection in the list pass the contact to the fragment. When the fragment is created start a new cursor loader and get all messages like you did with contacts in `MainFragment`. In the `onLoadFinished` method you can now append all cursor data to the second `TextView`.
- Add a `EditText` and a `Button` to the detail fragment when **your** contact is selected to send new messages to **your** feed. Save the message into the content provider when the button is pressed. This should update the messages.

Note: This application is missing a back-end. Next session we will introduce a MQTT server which is used as a message broker.