Ripple Report

version 1

Johannes Gårdsted Jørgensen - s093457

May 14, 2015

Contents

Introduction

Project start

This report documents the creation of an app for personal location sharing. I have often found myself in situations where i needed the knowledge of realtime update locations of other individuals. The cases could be a festival where i would be setting of scene stages. Often in that situation it is needed to know the location of a supervisor so you can evaluate if you can reach him within minutes or tens of minutes. Normally this would be done by sending text messages but i wanted something automatic you can just turn on and it will ping my location to my friends vice versa.

Ripple is an app that seeks to fulfill exactly this feature, by transmitting my location and showing my location and friends locations on a map.

Problem formulation

I want to create an app that lets people share their locations on a fixed frequency. Ripple has to be structured as a container activity with different fragments occupying the container view. Ripple will use the XMPP protocol for transmission of locations and it will use the Quickblox as a BAAS and xmpp server. Ripple will focus on decoupled identities in such a way that your Ripple logins are disposable and decoupled from your established personal accounts like Facebook etc. This will reduce the chance of being compromised as you can easily dump your account and create a new one. Ripple will seek to be secure by letting you decide which contacts you want to transmit to and a stretch goal is to implement Off-The-Record encryption on all xmpp traffic. Ripple shall plot all locations of enabled contacts on a map and all markers will be colored according to the time since their last location ping. If an enabled contact doesnt transmit pings for a certain period it will automatically be removed from the map for less cluttering.

Requirements

Name	Description
R1	Use quickblox BAAS sdk for XMPP and user account managing
R2	Use a container activity and fragments
R3	Option to dispose account and create new account
R4	Add contacts to contact list and enable/disable transmission to them
R5	Create the option to automatically zoom in and show all enabled contacts on map
R6	Show all contacts on a map, if a contact does not ping its location within the allowed time frame it will be removed from the map. Within the time frame a color change will represent the freshness of its location ping.
R7	Enable end to end encryption using Off-The-Record encryption

Process

I decided early on that i would make an app for location sharing.

I had been working with the version control system GIT before and decided to use it for this project.

I started by defining which subjects i would need to research in order to find the optimal solution before starting to implement them.

My main concerns from the start was:

- How to easily get XMPP up and running, best scenario would be without running the server myself
- Do i need a BAAS?
- How do i make the application secure?
- Should i put most weight on activities or fragments?

XMPP and BAAS

I started by researching which solutions existed for messaging between users. Originally i wanted my application to establish further security by having a dectralized protocol for communications.

However i soon figured out that decentralized communication is *bleeding edge* and in general not widely implemented. I dwelled at the extremely interesting *"Tele-Hash"* protocol by Jeremy Miller and wondered if i could make my own java implementation within the time fram but decided it was unrealistic. There is a short description of the Telehash protocol in the appendices.

If it shouldn't be a decentral protocol then the next best thing would be XMPP also originally invented by Jeremy Miller in the late 90's.

To create a XMPP system you need client implementations and a server. I tried to use the most recently updated XMPP libs for android i could find called *"Smack"*. I setup an xmpp server on my home server and tried to connect to it. After 2 days of tries and debugging i still had not established a connection.

When searching for a solution to one fo the endless non-descript XMPP exceptions from the smack driver i came across a reference to the Quickblox BAAS.

Quickblox used xmpp within but exposed a more simple xmpp api to the programmer.

I decided it was the way to go and a bonus was that they acted as the xmpp server as well so i didn't have to run it myself.

Secutity and authentication

When i had decided on the quickblox sdk i started looking for ways to implement security in the application. Security is critical since people are sharing their location and if a phone is compromised i wante to reduce the likelihood of the exposure of all the locations of said phones contacts.

I had some ideas to implement security:

Make accounts disposable and decoupled The idea is to let accounts in my system be as naked as possible and decouple them from peoples private information, like facebook, phone, email etc. In this way you obfuscate the user so that the context is implicit. What i mean is that if a phone is stolen and an unwanted entity gains access to the map view. This entity will only be able to see where a number of usernames are located and nothing more. So only if the thief/government has knowledge of the context of a user then the information gains value.

End to end encryption of all messages Early on a wanted the messages to furthermore be protected by encryption from end to end. Different options exist for this and they all have their specialities. I already knew about the PGP assymetric encryption but it would break my idea of disposable accounts since pgp is public/private key encryption and therefore a key pair would have to be generated and stored on each account creation. This of course is no problem programmatically but if a solution existed to to end-to-end encryption without the keypairs i would prefer this.

The solution seemed to be Off-The-Record encryption which was invented specifically for instant messaging. The OTR protocol uses a combination of many *"crypto"* tricks to generate secure encryption per session and per message.

In this way i could make sure a message was not intercepted by a man-in-the-middle attack and the location of a user could be exposed.

Sadly i came to this conclusion too late into the process and didn't have time to implement the OTR protocol. However i managed to do the needed research and figure out which OTR library would have been suitable.

Activities and fragments

Through the process i had implemented my app with a focus on activities since they were the first you learned to implement following the lessons. As i followed the Android lessons i was acquainted with the concept of fragments and how their lifecycle is dramatically alternative compared to the activity.

I rewrote the whole program to use fragments and decided to use a Singleton for storing information between fragments.

Theory

Quickblox is a powerful BAAS with sdk's for many different platforms with a focus on providing easy communication, including both push notifications and chat messaging, and easy user account management. Quickblox is a commercial BAAS with a free edition maxing out at 20 messages/second and 200000 monthly users. This makes it ideal for a student project.

Quickblox creates a nice streamlined sdk for android that hides away a lot of the complexity of the underlying libraries.

An example is that quickblox uses xmpp for its chat protocol and uses the *smack* android xmpp library but quickblox adds a level of abstraction by hiding all *smack* functions and leaving a cleaner more high level api for chat messaging.

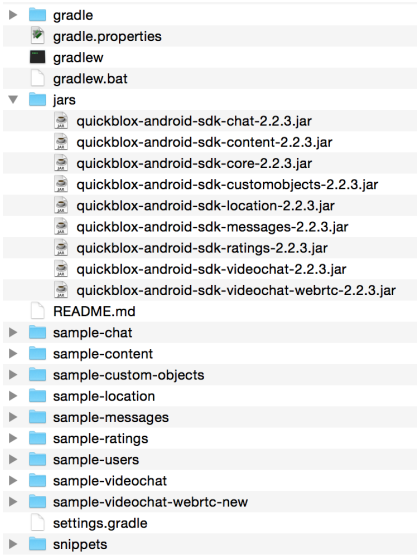
In this section i will try to explain the quickblox sdk centering around the User and Chat parts. I will cover implementation with examples and a guide on how to install the library.

The android sdk for quickblox in genereal presents you with a callback based asynchronous pack a functions. En example for what this means is that if you want to login you create a login try and a callback if login was succesfull. This relieves your own code from dealing with blocking api calls. See implementation/user for an example on this.

This section is inspired by the useful documentation provided by quickblox on their website [1]

Installation

You download the Quickblox android sdk from their website [1] as a zip with examples included. Below you can see how the downloaded sdk is organized:



The jars directory containts all the jars of the sdk, you choose which jars to embed in your project according to which functionality you want from quickblox.

You can always take a look in the different sample projects if you are uncertain which jars are needed for a certain part of the api.

Setup example

You need to tell cradle to include the quickblox jars when building.

As an example i want to use the Users and Chat apis form the quickblox sdk. The User api lets you create and authenticate users as well as let users manage a user profile. The Chat api lets users chat with eachother and check if another user is online.

For this example you would need the following jars: (i replaced the version with VERSION)

- quickblox-android-sdk-core_VERSION.jar
- quickblox-android-sdk-chat-VERSION.jar

The core jar contains all the core functionality of Quickblox like for example the User api. The chat jar contains all functions related to chat messaging.

You use the following syntax in your cradle settings

```
dependencies {
   compile files('libs/quickblox-android-sdk-core-2.1.jar')
   compile files('libs/quickblox-android-sdk-chat-2.1.jar')
}
```

And remember to define permissions for internet access in your manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Implementation examples

User

To use the Users api you need to start by creating a session.

Session

Quickblox provides a nice *createSession* function that takes a callback as a parameter. Therefore you don't have to worry about blocking the main thread!

```
QBAuth.createSession(new QBEntityCallbackImpl<QBSession>() {

    @Override
    public void onSuccess(QBSession session, Bundle params) {
        /*
        YEAH you created your first quickblox session!
        now go and have some quickblox fun
        */
    }

    @Override
    public void onError(List<String> errors) {
        /*
        Too bad, there was an error establishing contact to the api server
        try look in the errors list for an explanation!
        */
    }
});
```

The quickblox api expects you to implement some kind of state machine where the different callbacks place you in a different state. The createSession callback should lead either to a *session success* or *connection error* state.

Sign up

If you are in the *session success* state you are able to do api calls to quickblox. Lets start by creating a user:

Lets create a user with the following information:

- username = karlmarx
- password = kapital
- phone number = 11223344

```
final QBUser user = new QBUser("karlmarx", "kapital");
user.setPhone("11223344")

QBUsers.signUp(user, new QBEntityCallbackImpl<QBUser>() {
    @Override
```

```
    public void onSuccess(QBUser user, Bundle args) {
        /*
        YEAH! you chose a unique unused username and the api
        succesfully created a new user
        */
    }

    @Override
    public void onError(List<String> errors) {
        /*
        Too bad, your new account were not accepted,
        there can be any number of reasons, try look in the errors list ;-)
        */
    }
});
```

A quickblox user can have many more fields set on itself both at creation and later on. These fields include:

- facebook id
- twitter id
- email
- tags (as a list of strings)
- website url

Sign in

When you have succesfully signed up you are allowed to sign in using the created user. You can sign in using a number of ways ranging from twitter/facebook tokens to using the native quickblox users api.

Continuing on our example i will describe the process of logging in with a username and a password.

```
QBUser user = new QBUser("karlmarx", "kapital")

QBUsers.signIn(user, new QBEntityCallbackImpl<QBUser>() {
    @Override
    public void onSuccess(QBUser user, Bundle params) {
        /*
        Yeah you succesfully logged in!
        */
    }

    @Override
    public void onError(List<String> errors) {
        /*
        Too bad either your credentials were rejected or any other number of reasons
        look in the errors list for forensics ;-)
        */
    }
});
```

This concludes the section on how to establish a quickblox session, next up is sending a *hello world* chat message.

Chat

This section takes for granted that you have an authenticated session established. To begin chatting you need to establish some formalia beforehand. These formalia include the ones required by the xmpp protocol. More specifically you need to tell the xmpp protocol which frequency it will send an *"im online"* presence notification to keep you regarded as online. This notification is part of the xmpp protocol and is not a traditional *"push notification"*.

You do it like this:

```
if (!QBChatService.isInitialized()) {
    QBChatService.init(context);
}
QBChatService.getInstance().startAutoSendPresence(60);
```

Here we initialize the chatservice if its not allready initialized and then start transmitting presence notifications to quickblox. If you want to handle changes in the connection you have to implement the *"ConnectionListener"* interface.

Chat "hello world"

Two ways to chat exist, 1-1 and group chat. I will describe 1-1 chat since it does not need the establishement of a group room beforehand.

To start a chat with another user you need to know the id of the user. If you don't know the id of the user you can get it by using another known field of the user.

Here is an example of how to aqcuire the id of a user with username *"karlmarx"*:

```
QBUsers.getUserByLogin("karlmarx", new QBEntityCallbackImpl<QBUser>() {
    @Override
    public void onSuccess(QBUser user, Bundle args) {
        int user_id_of_karl_marx = user.getId()
    }

    @Override
    public void onError(List<String> errors) {
        /*
        Too bad you have not supplied right info, check errors list for explanations!
        */
    }
});
```

When you have the id of the user, then you are able to create a chat with this user.

It works like this:

Define a QBMessageListener of type QBPrivateChat

```
QBMessageListener<QBPrivateChat> privateChatMessageListener = new QBMessageListener<QBPrivat
    @Override
    public void processMessage(QBPrivateChat privateChat, final QBChatMessage chatMessage) {

    }

    @Override
    public void processError(QBPrivateChat privateChat, QBChatException error, QBChatMessage

    }

    @Override
    public void processMessageDelivered(QBPrivateChat privateChat, String messageID){

    }

    @Override
    public void processMessageRead(QBPrivateChat privateChat, String messageID){
    }
};
```

Define a QBPrivateChatManagerListener

```
QBPrivateChatManagerListener privateChatManagerListener = new QBPrivateChatManagerListener()
    @Override
```

```
    public void chatCreated(final QBPrivateChat privateChat, final boolean createdLocally) {
        if(!createdLocally){
            privateChat.addMessageListener(privateChatMessageListener);
        }
    }
};
```

Add the QBPrivateChatManagerListener to the QBChatService

```
QBChatService.getInstance().getPrivateChatManager().addPrivateChatManagerListener(privateCha
```

Create a QBChatMessage and send it

```
Integer opponentId = user_id_of_karl_marx;

try {
    QBChatMessage chatMessage = new QBChatMessage();
    chatMessage.setBody("Hello world");

    privateChat = privateChatManager.createChat(opponentId, privateChatMessageListener);
    privateChat.sendMessage(chatMessage);
} catch (XMPPException e) {

} catch (SmackException.NotConnectedException e) {

}
```

The exceptions can be quite non descriptive since they often refer to functions from inside the sdk jars.

This concludes the theory on the quickblox android sdk

1(1, 2) http://quickblox.com/developers/Android#Download_Android_SDK.

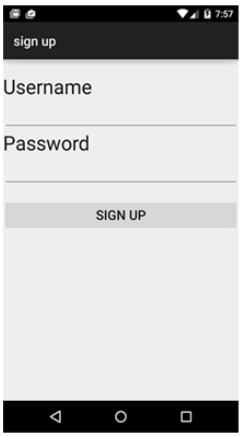
Design

Ripple consist of variations of four view:

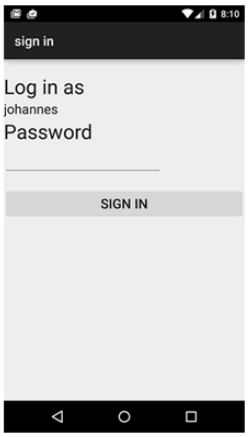
- Sign up
- Sign in
- Map
- Contacts list

Underneath you see the different views and a few variations on the *Map* view.

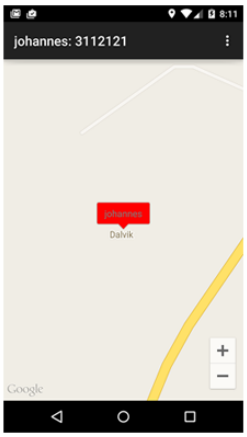
[A] - [SIGN UP]



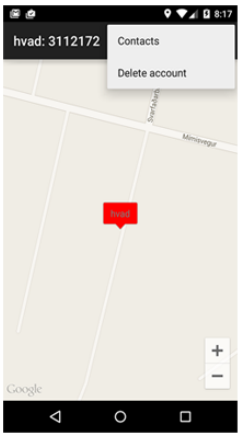
[B] - [SIGN IN]

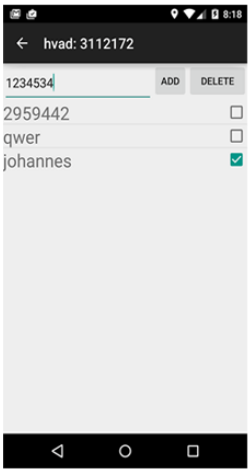


[C] - [MAP VIEW]



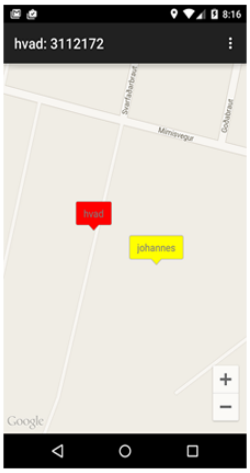
[D] - [MAP VIEW]





[D] - [CONTACTS]

[E] - [MAP VIEW]



[F] - [MAP VIEW]

At first i was experimenting with auto generated usernames to further obfuscate them as disposable and give people the impression that they were *"non-personal"* but it was hard to implement the functionality of figuring out which usernames were not taken at the BAAS so i went with people typing in usernames when signing up *(see Figure A)*

Markers

Letting people customize their usernames also gave me the option to use these to personalize the map markers. I decided to set the title of the container activity to a combination of username and user_id so it is easier when you are adding contacts (you just look at their screen and type in their user_id and press add)

The map markers change color as a function of the time-since-last-ping. In figure E and F you can see the functionality of the color change. We see in figure E that user *"hvad"* has added and enabled the user *"johannes"*. Johannes is green because *"hvad"* has just received a location update from *"johannes"*.

In figure F we see that *"johannes"* has stopped sending location updates to *"hvad"*, this has changed the color of the *"johannes"* marker into yellow, the step between *"healthy"* and *"removed from map*".

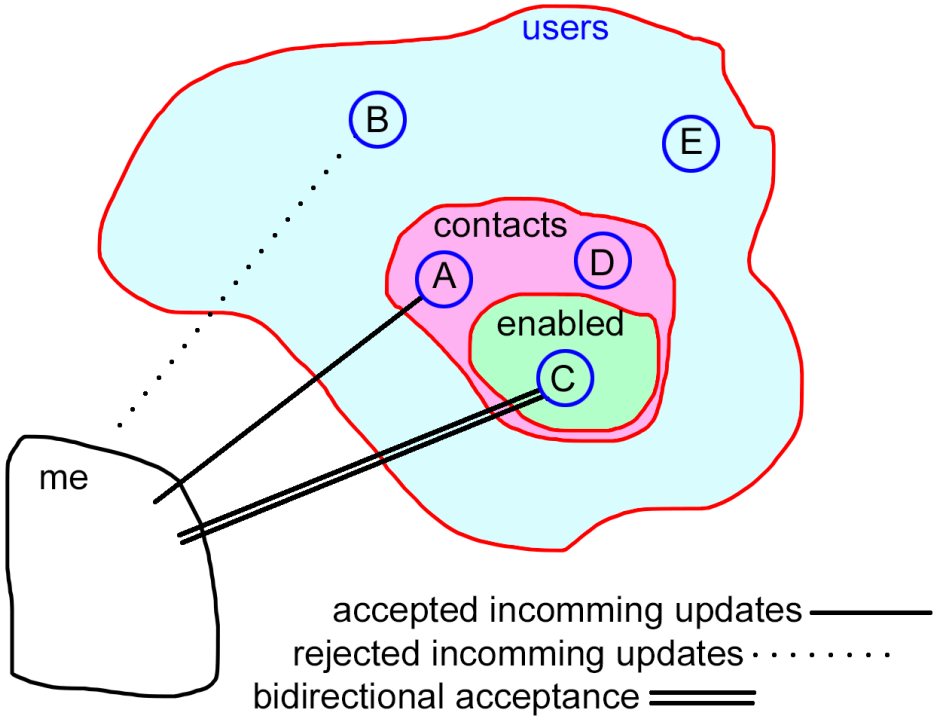
Contacts

Figure D shows the contacts view. This view is accessed from the action bar menu shown figure D. The contacts view lets you add a new contact, presently by typing in the user id of the contact. The view also lets you manage which contacts are enabled.

The contact list describes which users you accept location updates from, like a whitelist. It also features a toggle switch to enable the contact.

Enabling a contact means that you add the contact to the subset of contacts you transmit your location to.

The figure beneath describes the relationships:



1 : *accepted incomming updates:*

I accept, and draw markers for, incomming location updates from all contacts.

2 : *rejected incomming updates:*

I reject and ignore incomming location updates from users not in contact sub-set.

3: bidirectional acceptance:

The same as *1* with the added functionality that i actively transmit my location to users in this sub-sub-set at a fixed frequency.

Implementation

Persistence

Ripple uses a number of different tools to provide persistence of data to the user. Ripple uses a singleton to store information of currently signed-up user as well as an interface to load and save a list of options to the preference manager.

Indices and tables

- `genindex`
- `modindex`
- `search`