Klottr: Developer Manual

Latest revision: 2013-10-15

A quick-start guide and overlook of the Klottr Android application.

Getting started

git clone https://github.com/egenvall/projektht13/(Klottr.git)

Dependencies

- Java 7 SE development enviroment
- Android SDK
- A (virtual) Android device

Android SDK targets

- Minimum SDK: **10**
- Target SDK: **17**

Building and installing

To build and install Klottr.apk on a connected Android device simply run the project in Eclipse.
In this process you will be promtped to choose wich device to run it with.

Release procedure

This section describes the steps taken before every major release of the Klottr application.

Requirements

To build an application package in release mode, it needs to be signed with a certificate. There is a step by step guide in http://developer.android.com/tools/publishing/appsigning.html where you will learn how to sign applications for release.
You also need a keystore in your system. The path of your keystore should be specified in an project.properties file in the project directory.

Building a release package

Use Eclipse to build an .apk file for release:

Open the project in Eclipse and click File > Export then choose Android/Export Android Application,
In this step you will also be prompted for the passphrase of your keystore.

Release requirements

Every release's directory include the following:

- An application package (see above).
- A document for release notes with the following headings
    - New features
    - Changed features
    - Removed featus
    - Known bugs (refer to bug ID)
    - Coming features

Tests

The tests for Klottr are included in a seperate folder called Projektht13Test. To run the included tests you need to add the Projektht13 project in to Projectht13Test build path. To to that right click Projectht13Test > Properties > Java Build Path
Then press the projects tab and add Projectht13 folder and click ok.

Definition of done

Consider a completed user story resulting with unexpected bug/bugs in another feature as done, But the bugs should be filed.

Every story **MUST** do the following before marked as "finished":
- Match the specifications of the title and description in the story.
- Include relevant tests, if applicable
- Include relevant documentation, if necessary
- Been tested manually with an android device, not with emulator

Every story **SHOULD** do the following before marked as "finished":
- Include efficient code, based on application performance

Architecture

The application code is divided in packages and is organized by its (job), like all the classes that are

activities(Android Activities) are in one package, for data provider interactions(getting or storing messagepoints/posts) etc.

Content Provider

The database is accessed through http requests to either of four php files located at http://web.student.chalmers.se/~wajohan/mongodb/

.

getmessages.php returns by default all messages in the database, formatted as JSON, but it can be told to filter messages that are older than a certain time or belongs to a certain MessagePoint. E.g to get all messages belonging to messagepoint with id nr 420 that are older than 1381578304 (seconds after 1970) we request the url

- http://web.student.chalmers.se/~wajohan/mongodb/getmessages.php?mpid=420&after=1381578304
- getmessagepoints.php returns a list of all the messagepoints in JSON format. There are no
- postmessagepoint.php recieves a messagepoint, the data is transmitted through the POST variables "mpid" (id of MessagPoint), "x" and "y" (longitud and latitude).

- postmessage.php recieves a message. The data is transmitted with the POST variables "mpid", "name" and "message"

Implementation

In MapModel.java there are three public methods associated with connection to the database:

updateMPs() initiates an AsyncTask that issues two http requests, the first to fetch all MessagePoint objects and the second to fetch all Messages.

AddMessagePoint(Point position) First updates the local messagePoints ArrayList and then initiates an AsyncTask that performs the necessary http connection to the database

AddMessageToMessagsePoint(int id,Message message) Works in a similar manner as AddMessagePoint, it first updates the local MessagePoint and then initiates an AsyncTask that posts the message to the server.

Performance discussion

For this app there is a requirement to keep things as synchronized to the central database as possible. For example if someone puts up a messagepoint and writes something the other clients running the app should notice this without too much delay. At the moment this is solved by calling updateMP's every 1337 milliseconds.

The reader who are experienced in the field of programming and network programming may realize the inefficiency of current usage of the method updateMPs(). The programming is done according to the KISS principle and the presumption that this application won't reach world-wide popularity and the network load on servers is not an issue. However, for any continous developement on this project there exists some ideas which could enhance efficiency and minimize network load:

- The client side should be modified in order to ONLY download MessagePoints within a certain area.

Usually the user is focused on a very small part of the world, there is no need for the application to keep track of and download every MessagePoint in the whole wide world. The server side should be modified to support area limited requests as well.

- The client should only fetch messages belonging to certain MessagePoints, possibly just the one that the user at the moment has opened. Another option is to only fetch those MessagePoints that are in the visible area. The latter would have the advantage of minimizing experienced delay by prefetching every MessagePoint that the user might open.

Google API connection

For the application to get access to the Google map service a special key is required. This key is unique for every app and bound to certain keystores. The keys can be generate d by registering at google api console.

Notifications and Services

No notifications and services areimplemented.

Database documentation

Due to our common aversion against having a running sql server 24/7 in our own homes, and due to the fact that chalmers IT helpdesk could not help us with our IT needs we decided to use chalmers php capable web servers to write our own little "database".

Storage

The database stores data in two text files, messages.txt and messagepoints.txt. The data is stored with each post is comprised of three rows, then a blank row and then the next post.
The format for messagepoints is:
<MessagePoint ID>
<latitude>
<longitude>
(empty row)

The format for messages is:
<MessagePoint id>
<name/author>
<message>
<seconds since 1970>
(empty row)

HttpInterface

The http interface consists of four php files. They are:
getmessagepoint.php
getmessages.php
postmessagepoint.php
postmessage.php

The names are self explanatory. The programming is in low level manner and is not intended to be practical

to extend. For any real application of this app the idea is that a “real” database(mysql, postgresql) should be used.