Hochschule für Technik und Wirtschaft Berlin
- Campus Wilhelminenhof -

Belegarbeit

**im Studiengang Angewandte Informatik -
Schwerpunkt Mobile Anwendungen**

Thema: Android Applikation für Darstellung von Daten

Autor: Marcel Ebert
MatNr. S0558606

Version vom: 4. Januar 2018

Betreuer: Prof. Dr. Alexander Huhn

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

Activity .... Bildet das Kernstück einer Android-Anwendung und steuert die Benutzerinteraktion

Broker ..... Bezeichnung für den Server in einem MQTT-Netzwerk

Fragment .. Ein in Android-Activites verwendeter Teil einer Benutzeroberfläche

Topic ..... Eine Zeichenkette, welche vom Broker verwendet wird um Nachrichten zu filtern

1 Einleitung

Im Rahmen dieses Projektes soll eine Anwendung für die Android-Plattform entwickelt werden, welche dem Benutzer die Möglichkeit gibt, sich mit einem Server zu verbinden um Nachrichten, welche in dem Netzwerk des Servers verschickt werden, zu empfangen. Der Inhalt dieser Nachrichten soll persistent gespeichert und graphisch dargestellt werden können. Dies soll dem Benutzer der Anwendung die Möglichkeit geben, Daten eines Netzwerks in Echtzeit graphisch aufgearbeitet zu bekommen und ein besseres Verständnis zu erhalten. Es wird vorausgesetzt, dass das Netzwerk des Servers MQTT als Nachrichtenprotokoll verwendet. Da die Anwendung für die Android-Plattform entwickelt werden soll, wird Android-Studio als Entwicklungsumgebung und Java als Programmiersprache verwendet.

Der Quelltext ist zu finden auf: https://github.com/Marcel-Ebert/MQTTDataVisualizer.

2 Grundlagen

2.1 MQTT

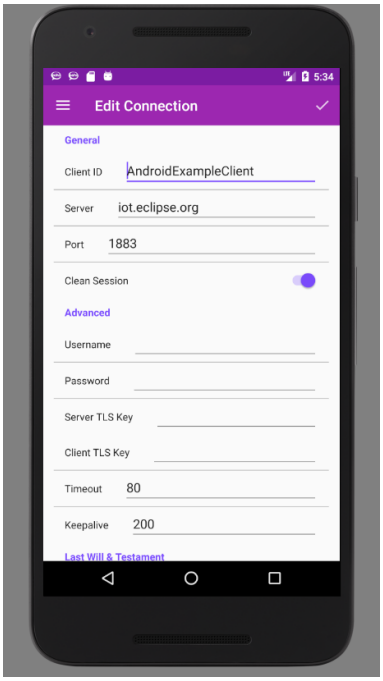
MQTT steht für 'Message Queuing Telemetry Transport' und ist ein Nachrichtenprotokoll für Machine-to-Machine-Kommunikation. Es ist 'publish-subscribe'-basiert und soll die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten, trotz hoher Verzögerungen oder beschränkter Netzwerke, ermöglichen. MQTT ist ein Client-Server-Protokoll, wobei der Server auch 'Broker' genannt wird. Wenn ein Client die Verbindung zu einem Broker aufgebaut hat, kann er diesem Nachrichten schicken, welche aus einem Topic und einem Nachrichteninhalt bestehen. Diese Nachricht wird dann an alle Clients geschickt, die dem Broker mitgeteilt haben, dass sie diesen Topic abonnieren wollen. Weiterhin besitzen Nachrichten einen von drei möglichen Quality-of-Service Parametern: 0, die Nachricht wird einmal gesendet und kommt möglicherweise nicht an; 1, die Nachricht wird so lange gesendet, bis der Empfang bestätigt wird und kann beim Empfänger mehrfach ankommen oder 3, es wird sichergestellt, dass die Nachricht genau einmal bei den Empfängern ankommt.[BG14]

2.2 Sampleanwendung

Die entwickelte Anwendung verwendet als Grundlage eine Sampleanwendung, welche zum 'Eclipse Paho'-Projekt gehört[1]. Dieses Sample wurde verändert und erweitert um die neuen Ansprüche zu erfüllen. Das 'Eclipse Paho'-Projekt ist entstanden um quelloffene Clientimplementationen von MQTT und MQTT-SN Nachrichtenprotokolle zur

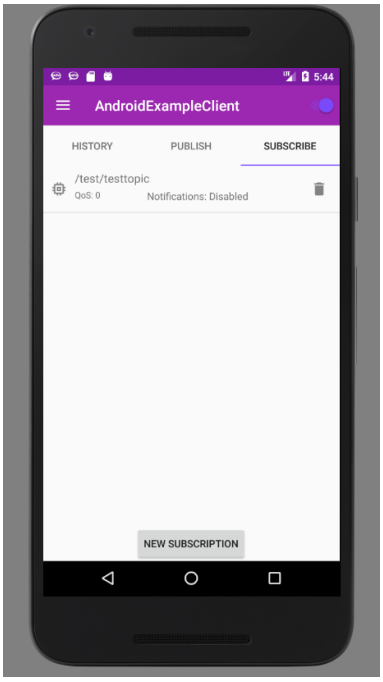
[1]Link zu Github von Sampleanwendung: https://github.com/eclipse/paho.mqtt.android

Verfügung zu stellen.[2] Es gibt Bibliotheken für 'C', 'Java', 'JavaScript', 'Python' und noch weitere.

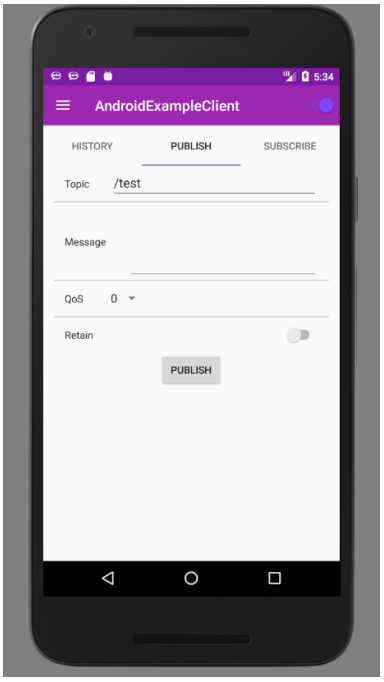


(a) Anlegen einer neuen Verbindung

(b) Navigationsleiste



(c) Subscribe-Tab



(d) Publish-Tab

Abbildung 1: Sampleanwendung

Die Sampleanwendung benutzt einen 'Navigationdrawer', also eine Seitennavigationsleiste (siehe Abbildung 1a), um eine bessere Navigation zu ermöglichen. Aus dieser Entwurfsentscheidung folgt, dass die gesamte Anwendung nur eine 'Activity' besitzt, da der Navigationdrawer dazu eingesetzt wird um das aktuelle Fragment der Activity

[2]Link zu Projektseite: https://www.eclipse.org/paho/

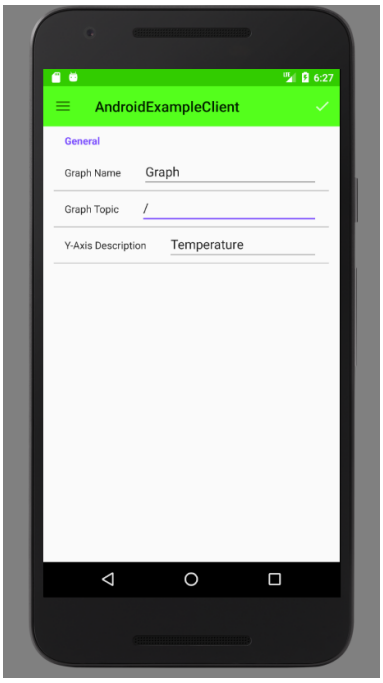
durch neue Fragmente zu ersetzen. Wenn der Navigationdrawer stattdessen neue Aktivitäten aufrufen würde, würde dies einen negativen Einfluss auf die Navigation in der Anwendung haben. Durch einen Klick auf das 'Add Connection'-Element in der Seitenleiste öffnet sich eine neue Ansicht, wo Informationen über die anzulegende Verbindung eingegeben werden müssen. Hierzu gehören der Name des Clients sowie die Adresse des Brokers und der Port, welcher benutzt werden soll. Danach erscheint die angelegte Verbindung im Navigationdrawer und durch Antippen dieser gelangt man zu der Hauptansicht der Verbindung. Diese Ansicht besteht aus drei Registerkarten: 'History', 'Publish' und 'Subscribe'. Außerdem gibt es in der oberen rechten Ecke einen Schieberegler, mit dem gesteuert werden kann, ob sich der Client mit dem Broker verbinden soll oder nicht. Im 'History'-Tab werden erhaltene Nachrichten gezeigt. Im 'Publish'-Tab hat man die Möglichkeit Nachrichten an den Broker in verschiedenen Topics zu schicken und im 'Subscribe'-Tab kann man bestimmte Topics abonnieren, deren Nachrichten dann im erwähnten 'History'-Tab angezeigt werden. Weiterhin öffnet sich durch langes Antippen einer Verbindung in der Seitenleiste eine neue Ansicht, wo man die Details der Verbindung bearbeiten kann.

3 Entwurf

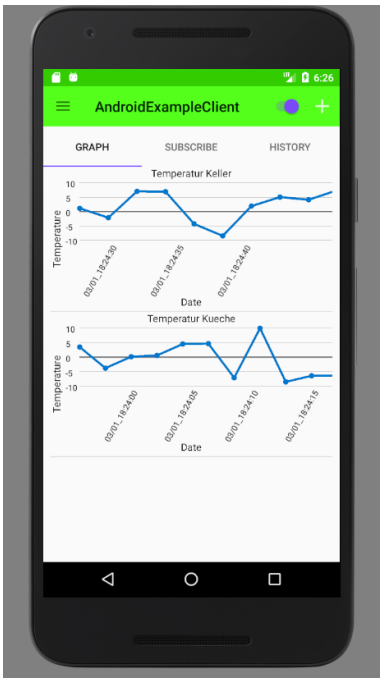
Die in den Grundlagen vorgestellte Sampleanwendung wird angepasst. Die Registerkarte 'Publish' wird entfernt, da die Anwendung nicht mehr dazu in der Lage sein muss Nachrichten im Netzwerk zu verschicken. Sattdessen wird eine Registerkarte 'Graph' eingefügt.

Wenn der Benutzer die Anwendung startet hat er die Möglichkeit bestehende Verbindungen zu betrachten, oder eine neue Verbindung anzulegen. Beim Anlegen einer neuen Verbindung, muss er die Adresse des Brokers sowie weitere Informationen über die Verbindung angeben. Danach wird versucht den Client mit dem Broker zu verbinden. Bei Erfolg wird der Benutzer zu einer anderen View geführt, die insgesamt drei Tabs hat: "Graph", "Subscribe" und "History".

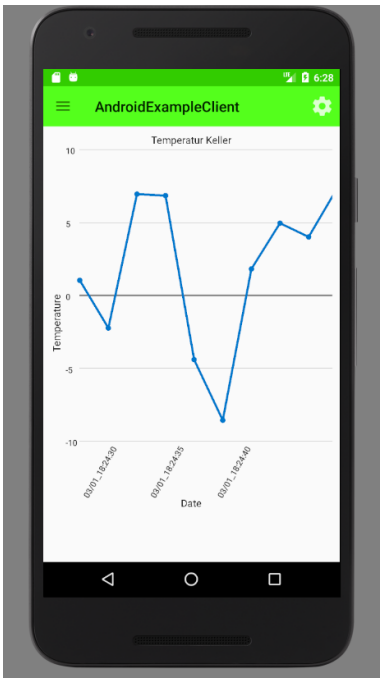
Im 'Graph'-Tab werden alle aktuell angelegten Graphen in einem 'Listview' dargestellt. Es lässt sich nach oben und unten scrollen, wenn der Platz nicht ausreicht um alle Graphen gleichzeitig darzustellen. Wenn der 'Graph'-Tab ausgewählt wurde gibt es oben rechts die Möglichkeit durch Klicken auf einen Button einen neuen Graphen anzulegen. Dazu müssen verschiedene Informationen wie der Name des Graphen, der zugehörige Topic, sowie die Beschreibung der y-Achse angegeben werden. Ein Graph kann immer nur die Daten von einem Topic darstellen. Im "Subscribe"-Tab können Topics abonniert und deabonniert werden. Im "History"-Tab werden Nachrichten angezeigt, die in der aktuellen Session vom Client empfangen werden. Die vom Broker geschickten Nachrichten, die zu den abonnierten Topics gehören, werden dem Benutzer angezeigt und persistiert.



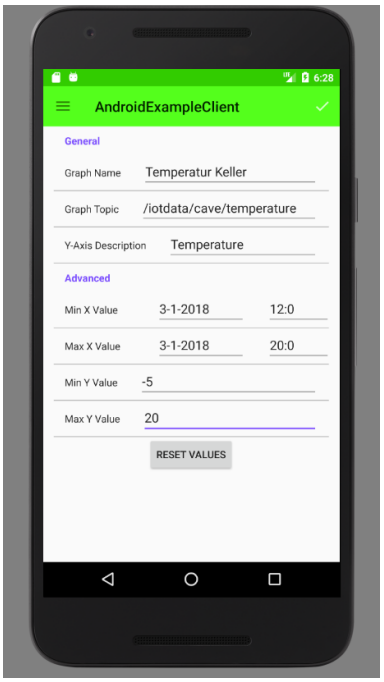
(a) Anlegen eines neuen Graphen



(b) Graph-Tab



(c) Detaillierte Ansicht eines Graphen



(d) Detaillierte Einstellungen eines Graphen

Abbildung 2: Graph-Fragmente

4 Implementierung

4.1 Empfangen von Nachrichten

Durch das 'MqttCallback'-Interface, welches im 'Eclipse-Paho'-Projekt definiert wurde, gibt es die Möglichkeit auf bestimmte Ereignisse zu reagieren. Diese Ereignisse sind das Verlieren der Verbindung zum Broker, das Eintreffen einer neuen Nachricht und

das erfolgreiche Versenden einer Nachricht. Dieses Interface wurde durch einen 'MqttCallbackHandler' implementiert und dieser leitet die eintreffenden Nachrichten an die entsprechende Verbindung weiter. Zu einer erstellten Verbindung lassen sich Listener hinzufügen, die die Nachricht dann überreicht bekommen. Das bedeutet, wenn der Client mit dem Broker verbunden ist und Nachrichten empfangen werden, dann erhalten diese Nachrichten einen Zeitstempel und werden an alle registrierten Listener weitergereicht. Die erstellten Graphen besitzen einen eigenen Listener für diese Nachrichten und überprüfen dann, ob der Inhalt graphisch dargestellt werden kann. Dafür gibt es zwei Möglichkeiten: Entweder die Nachricht besteht nur aus einer Fließkommazahl, oder sie besteht aus einem String gefolgt von einem Leerzeichen und einer Fließkommazahl. Im zweiten Fall wird dann die Nachricht aufgeteilt und die Fließkommazahl extrahiert. Diese wird dann dem Graphen hinzugefügt.

4.2 Implementierung des GraphView-Framework

Zur graphischen Darstellung der Daten wird das GraphView-Framework benutzt. GraphView ist eine quelloffene Bibliothek, welche die Möglichkeit bietet verschiedene Diagrammtypen zu erstellen und in Android-Anwendungen zu implementieren. Das Framework wird von Jonas Gehring entwickelt und gepflegt.[3] Es bietet Möglichkeiten um die Achsen zu beschriften, das Ansichtsfenster zu verändern, sodass nur bestimmte Zahlenbereiche sichtbar sind, Events zu erzeugen, wenn ein Datenpunkt angeklickt wird und vieles Mehr.

In diesem Projekt werden nur Liniengraphen verwendet. Die konkrete Implementierung zusammen mit verwendeten Eigenschaften kann in Listing 2 eingesehen werden. Durch Verwendung eines 'DateAsXAxisLabelFormatter' können dem Graphen auf der x-Achse Zeitdaten hinzugefügt werden. Das angezeigte Format wird durch eine Insatz eines 'DateFormat' bestimmt, welche dem 'DateAsXAxisLabelFormatter' übergeben wird.

In den erweiterten Einstellungen (siehe Abbildung 2d) ist es möglich den Bereich der x- und y-Achse einzuschränken. Durch Antippen der ersten 'EditText'-Box neben den Labels 'Min X Value' und 'Max X Value' öffnet sich ein 'DatePicker'-Dialog. Dieser wurde verwendet um die Eingabe eines gültigen Datumsformates zu vereinfachen. Wenn die zweite 'EditText'-Box neben den Labels angetippt wird erscheint ein 'TimePicker'-Dialog. Bei nicht vorhandenen oder gleichen Werten von 'Min X Value' und 'Max X Value' ergeben sich die Grenzen der x-Achse aus dem kleinsten und größten x-Wert der Datenpunkte. Analog dazu verhält es sich bei 'Min Y Value' und 'Max Y Value'. Wenn beide null sind oder den gleichen Wert haben entscheiden der kleinste und größte y-Wert der Datenpunkte. Wenn 'Min X Value' und 'Max X Value' so gewählt werden, dass nur ein Ausschnitt der vorhanden Datenpunkte angezeigt wird, gibt es die

[3] Projektseite: http://www.android-graphview.org/

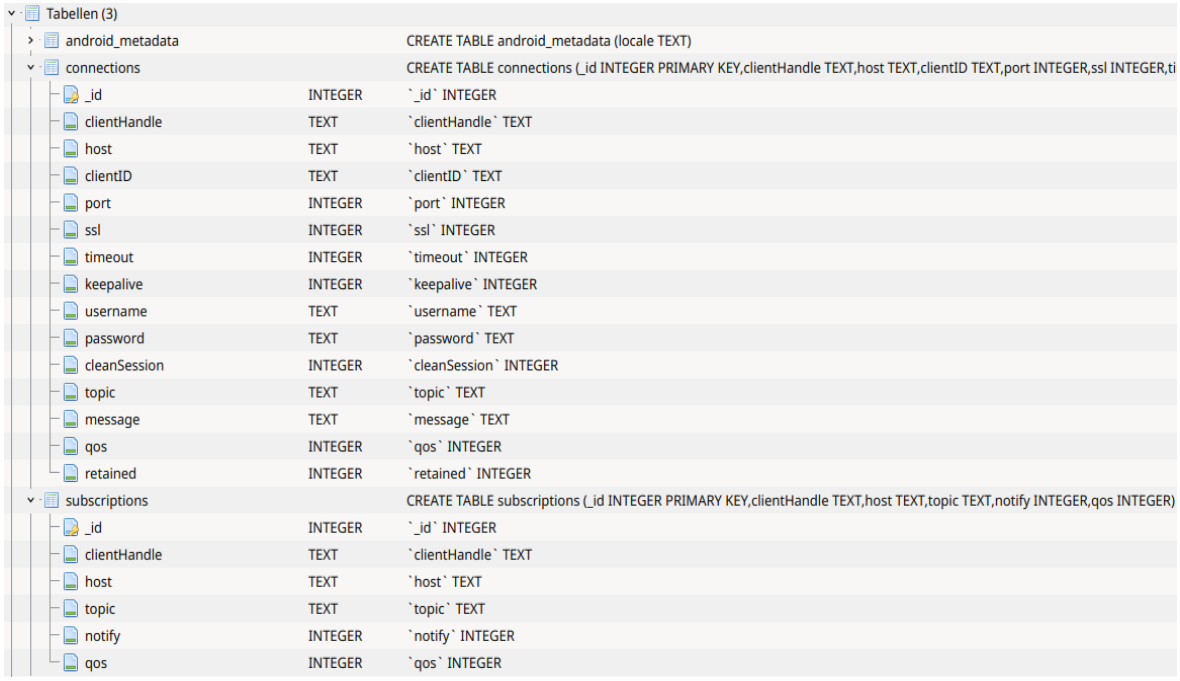
Möglichkeit durch die Ansicht zu scrollen, also das Sichtfenster nach links und rechts zu bewegen. Dafür müssen allerdings genügend Datenpunkte vorhanden sein. Wenn alle Datenpunkte in der aktuellen Ansicht angezeigt werden ist es nicht möglich zu scrollen.

Graphen können durch langes Tippen im 'Graph'-Tab (Abbildung 2b) entfernt werden. Dabei werden die persistierten Nachrichten die zum Topic des Graphen gehören nicht gelöscht. Diese werden erst entfernt wenn die Subscription gelöscht wird.

4.3 Persistente Speicherung

Zur persistenten Speicherung von Daten wird SQLite benutzt. SQLite ist eine freie, leichtgewichtige Implementierung einer SQL-Datenbank, wobei es keine vollständige Implementierung des SQL-Standards enthält. Für den Umfang dieses Projekts genügen die vorhanden Funktionen. Der Aufbau der Dateien wird im Folgenden erläutert.

Es gibt eine "connections.db"-Datei, in welcher Informationen über die verschiedenen Clients und ihre Eigenschaften gespeichert werden. Es gibt zwei Tabellen, eine für die Verbindungen und eine andere für die "Subscriptions". In der Verbindungstabelle werden unter anderem ein eindeutiger Clientname, der vom Benutzer gewählte Name sowie der Host, von dem Nachrichten empfangen werden sollen, zusammen mit der Portnummer, die für die Verbindung benutzt werden soll, gespeichert. In der anderen Tabelle werden für die vom Benutzer erstellten Verbindungen die zu abonnierenden Topics gespeichert.



Tabellen (3)		
android_metadata		CREATE TABLE android_metadata (locale TEXT)
connections		CREATE TABLE connections (_id INTEGER PRIMARY KEY,clientHandle TEXT,host TEXT,clientID TEXT,port INTEGER,ssl INTEGER,ti
_id	INTEGER	`_id` INTEGER
clientHandle	TEXT	`clientHandle` TEXT
host	TEXT	`host` TEXT
clientID	TEXT	`clientID` TEXT
port	INTEGER	`port` INTEGER
ssl	INTEGER	`ssl` INTEGER
timeout	INTEGER	`timeout` INTEGER
keepalive	INTEGER	`keepalive` INTEGER
username	TEXT	`username` TEXT
password	TEXT	`password` TEXT
cleanSession	INTEGER	`cleanSession` INTEGER
topic	TEXT	`topic` TEXT
message	TEXT	`message` TEXT
qos	INTEGER	`qos` INTEGER
retained	INTEGER	`retained` INTEGER
subscriptions		CREATE TABLE subscriptions (_id INTEGER PRIMARY KEY,clientHandle TEXT,host TEXT,topic TEXT,notify INTEGER,qos INTEGER)
_id	INTEGER	`_id` INTEGER
clientHandle	TEXT	`clientHandle` TEXT
host	TEXT	`host` TEXT
topic	TEXT	`topic` TEXT
notify	INTEGER	`notify` INTEGER
qos	INTEGER	`qos` INTEGER

(a) Aufbau der Tabellen in "connections.db"

_id	clientHandle	host	clientID	port	ssl	timeout	keepalive
Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern
1	78O6M9NT-...	iot.eclipse....	TestClient1	1883	0	80	200
2	RKUIJA3Y-io...	iot.eclipse....	TestClient2	1883	0	80	200

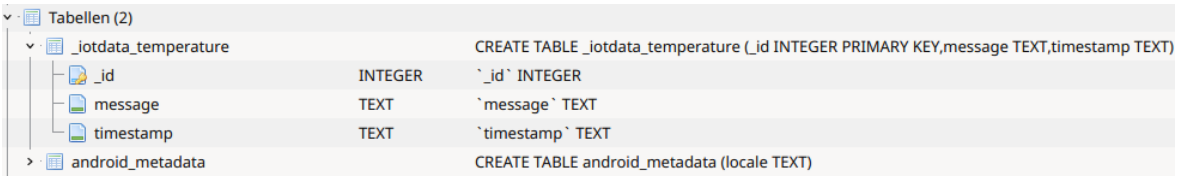
(b) Inhalt der Connections-Tabelle

_id	clientHandle	host	topic	notify	qos
Filtern	Filtern	Filtern	Filtern	Filtern	Filtern
1	RKUIJA3Y-iot.eclipse.org-TestClient2	NULL	/iotdata/temperature	0	0
2	78O6M9NT-iot.eclipse.org-TestClient1	NULL	/iotdata/temperature	0	0

(c) Inhalt der Subscriptions-Tabelle

Abbildung 3: connections.db

Weiterhin gibt es für jeden Client eine Datei "<ClientID>.db", in welcher alle empfangenen Nachrichten zusammen mit ihrem Zeistempel gespeichert werden. Es wird in der Datenbank für jedes abonnierte Topic eine eigene Tabelle angelegt, wobei sich der Name aus dem Topic-Namen mit "/" ersetzt durch "_" ergibt.



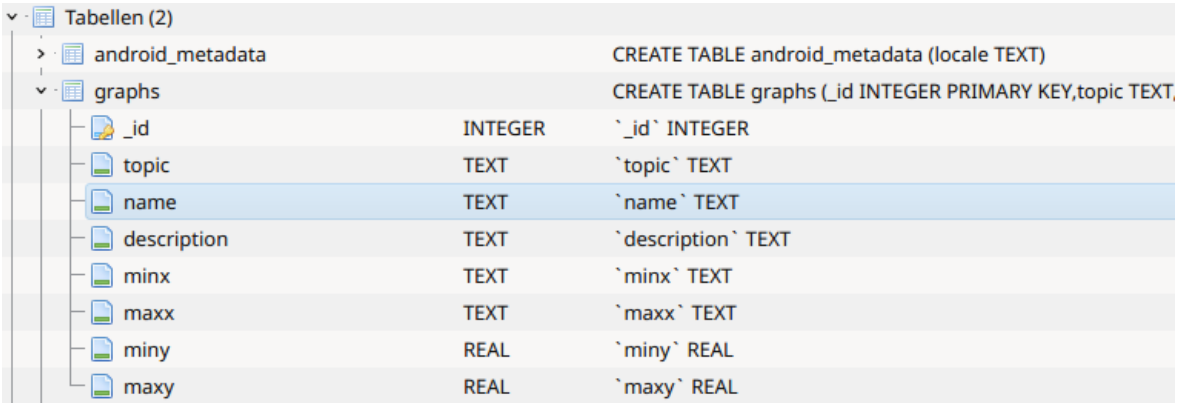
(a) Aufbau der Message-Tabelle

_id	message	timestamp
Filtern	Filtern	Filtern
1	-2.0101251408732175	2017-12-364 16:04:38
2	-6.5348021705888275	2017-12-364 16:04:40
3	-4.282726095185803	2017-12-364 16:04:42
4	-9.020907004924386	2017-12-364 16:04:44
5	-9.779592679661736	2017-12-364 16:04:47
6	0.7527580719780058	2017-12-364 16:04:49
7	6.693215201568435	2017-12-364 16:04:51
8	3.5519397567984363	2017-12-364 16:04:53
9	1.5054792808735424	2017-12-364 16:04:55
10	4.768400193407613	2017-12-364 16:04:57
11	-6.534360734518905	2017-12-364 16:04:59
12	8.996461365287956	2017-12-364 16:05:02
13	4.182993803306282	2017-12-364 16:05:04
14	-9.909888591174942	2017-12-365 17:04:38
15	-1.9000291785798158	2017-12-365 17:04:40
16	-5.446845140114631	2017-12-365 17:04:42
17	-1.8543043617321846	2017-12-365 17:04:44
18	-9.68619598304742	2017-12-365 17:04:46

(b) Inhalt der Message-Tabelle

Abbildung 4: <ClientID>.db

Schließlich wird auch noch für die anzulegenden Graphen eine Datenbank-Datei erzeugt. Diese wird "<ClientID>_graph.db" genannt. Dort werden Informationen zu erstellten Graphen und deren Beschreibung, sowie Minimum und Maximum auf x- und y-Achse gespeichert.



(a) Aufbau der Graph-Tabelle

_id	topic	name	description	minx	maxx	miny	maxy
Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern
2	/iotdata/kitchen/temperature	Temperatur Küche	Temperature	2018-01-04 12:00...	2018-01-04 20:00:...	0.0	30.0
3	/iotdata/basement/temperature	Temperatur Keller	Temperature	0	0	0.0	0.0

(b) Inhalt der Graph-Tabelle

Abbildung 5: <ClientID>_graph.db

5 Bedienung

Nach Starten der App muss zunächst eine Verbindung zu einem Broker festgelegt werden. Dazu öffnet man die Seitenleiste und tippt auf 'Add Connection'. Nach Anlegen der Verbindung sollte man zunächst im "Subscriptions"-Tab die Topics abonnieren, zu denen man einen Graphen anlegen möchte. Um in diesen Tab wechseln zu können muss man den Client zuerst mit dem Broker verbinden, also oben rechts den Schiebeschalter betätigen. Danach kann man zurück in den "Graph"-Tab wechseln und oben rechts auf das weiße "+" klicken. Es öffnet sich eine neue Sicht, wo man Informationen über den Graphen angeben kann. Anschließend drückt man auf das weiße Häkchen oben rechts und der Graph erscheint nun im "Graph"-Tab. Eingehende Nachrichten für diesen Topic, welche nur eine Fließkommazahl oder einen String gefolgt von Leerzeichen und Fließkommazahl als Inhalt haben, werden sofort im Graph dargestellt. Für eine noch detailliertere Ansicht mit veränderbaren Achsen muss man auf das Graph-Objekt tippen. In der neuen Sicht kann man oben rechts auf das Zahnrad tippen um die erweiterten Optionen anzuzeigen.

6 Fazit

Ziel dieses Projektes ist es gewesen, eine Android-Anwendung zu entwickeln, welche Daten aus einem MQTT-Netzwerk empfangen, persistieren und graphisch darstellen kann. Dafür war es wichtig, geeignete Bibliotheken und Frameworks zu benutzen um die geforderten Funktionen effektiv implementieren zu können.

Die Sample-Anwendung hat Einblicke, in die professionelle Entwicklung einer Android-Applikation, ermöglicht. Es konnten Erfahrungen mit bestimmten Design-Mustern, wie zum Beispiel dem effektiven Austauschen von Informationen zwischen Fragmenten durch 'Bundles', gesammelt werden.

Durch Einsatz der GraphView-Bibliothek wurde eine Möglichkeit gefunden um die gewünschten Graphen einfach und trotzdem mit vielen verschiedenen Optionen zu implementieren. Es gibt an dieser Stelle noch diverse Verbesserungsmöglichkeiten. Zum Beispiel könnten Funktionalitäten implementiert werden, welche es ermöglichen in einem GraphView-Objekt verschiedene Liniengraphen darzustellen, welche zu verschiedenen Topics gehören. So könnte man durch Kombinieren von diversen Temperatur-Topics eines Netzwerks ein Temperatur-Diagramm für einen ganzen Bereich erstellen, sodass bessere und schnellere Vergleiche gemacht werden könnten. Die Funktion mehrere Liniengraphen in einem Graphview darzustellen wird von der GraphView-Bibliothek unterstützt.

Das Persistieren der Daten wurde mit SQLite erfolgreich umgesetzt, wobei die Verknüpfung der Dateien untereinander anders aufgebaut worden können. Es hätte auf die Erstellung mehrerer gleichartiger Dateien verzichtet werden können, wenn der 'Clienthandle' in die anderen Tabellen mitaufgenommen worden wäre. Aus Gründen der Einfachheit wurde hierauf verzichtet.

Es fehlt die Implementierung eines Fragmentes, welches Informationen über das Projekt, sowie die anderen verwendeten Ressourcen enthält. Im Moment existiert noch das 'Help & Feedback'-Fragment der Sampleanwendung, welches auf die 'Eclipse Paho'-Projektseite verweist.

Außerdem wären Verbesserungen der Benutzeroberfläche in der detaillierten Graphansicht möglich. Die Implementierung von Buttons, welche das schnelle Filtern von Datenpunkten ermöglichen, wäre sinnvoll, wie zum Beispiel Buttons, welche die Anzeige sofort anpassen, sodass die letzten 24 Stunden, die letzten drei Tage oder die letzte Woche angezeigt werden.

Ergebnisformulierend ist festzustellen, dass dieses Projekt gezeigt hat, dass die Entwicklung einer solchen Android-Anwendung zufriedenstellend möglich war, wobei es an einigen Stellen Möglichkeiten zur Verbesserung gibt.

Literaturverzeichnis

[BG14] Banks, Andrew ; Gupta, Rahul: *MQTT Version 3.1.1.* http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 2014

Anhang

```
1  /*******************************************************************************
2   * Copyright (c) 2009, 2014 IBM Corp.
3   *
4   * All rights reserved. This program and the accompanying materials
5   * are made available under the terms of the Eclipse Public License v1.0
6   * and Eclipse Distribution License v1.0 which accompany this distribution.
7   *
8   * The Eclipse Public License is available at
9   *    http://www.eclipse.org/legal/epl-v10.html
10  * and the Eclipse Distribution License is available at
11  *   http://www.eclipse.org/org/documents/edl-v10.php.
12  *
13  * Contributors:
14  *    Dave Locke - initial API and implementation
15  *               and/or initial documentation
16  */
17 package org.eclipse.paho.client.mqttv3;
18
19
20 /**
21  * Enables an application to be notified when asynchronous
22  * events related to the client occur.
23  * Classes implementing this interface
24  * can be registered on both types of client:
25  {@link IMqttClient#setCallback(MqttCallback)}
26  * and {@link IMqttAsyncClient#setCallback(MqttCallback)}
27  */
28 public interface MqttCallback {
29         /**
30          * This method is called when the connection to the server is lost.
31          *
32          * @param cause the reason behind the loss of connection.
33          */
34         public void connectionLost(Throwable cause);
35
36         /**
37          * This method is called when a message arrives from the server.
38          *
39          * <p>
40          * This method is invoked synchronously by the MQTT client. An
41          * acknowledgment is not sent back to the server until this
42          * method returns cleanly.</p>
43          * <p>
44          * If an implementation of this method throws an <code>Exception
45          * </code>, then the client will be shut down.
46          * When the client is next re-connected,
```

```
47              * any QoS 1 or 2 messages will be redelivered by the server.</p>
48              * <p>
49              * Any additional messages which arrive while an
50              * implementation of this method is running, will build up
51              * in memory, and will then back up on the network.</p>
52              * <p>
53              * If an application needs to persist data, then it
54              * should ensure the data is persisted prior to returning
55              * from this method, as after returning from this method,
56              * the message is considered to have been delivered,
57              * and will not be reproducible.</p>
58              * <p>
59              * It is possible to send a new message within an implementation
60              * of this callback (for example, a response to this message),
61              * but the implementation must not disconnect the client,
62              * as it will be impossible to send an acknowledgment for
63              * the message being processed, and a deadlock will occur.</p>
64              *
65              * @param topic name of the topic on the message was published to
66              * @param message the actual message.
67              * @throws Exception if a terminal error has occurred, and the
68              * client should be shut down.
69              */
70             public void messageArrived(String topic, MqttMessage message)
71                     throws Exception;
72
73             /**
74              * Called when delivery for a message has been completed, and all
75              * acknowledgments have been received. For QoS 0 messages it is
76              * called once the message has been handed to the network for
77              * delivery. For QoS 1 it is called when PUBACK is received and
78              * for QoS 2 when PUBCOMP is received. The token will be the same
79              * token as that returned when the message was published.
80              *
81              * @param token the delivery token associated with the message.
82              */
83             public void deliveryComplete(IMqttDeliveryToken token);
84
85 }
```

Listing 1: Das Interface 'MqttCallback'

```
1
2  /**
3   * Holds a graphview to display the graph in more detail
4   * has an options menu where attributes of the graph object can be edited
5   */
6  public class DetailedGraphFragment extends Fragment {
7
8      private Connection connection;
9
10     private Graph graph;
11     private GraphView graphView;
12     private IReceivedMessageListener listener;
13
14     public DetailedGraphFragment() {
15         setHasOptionsMenu(true);
16     }
17
18     @Override
19     public void onCreate(@Nullable Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         Map<String, Connection> connections = Connections
22                         .getInstance(this.getActivity()).getConnections();
23         connection = connections.get(this.getArguments()
24                         .getString(ActivityConstants.CONNECTION_KEY));
25         graph = (Graph) this.getArguments().get(ActivityConstants.GRAPH_KEY);
26
27         listener = new IReceivedMessageListener() {
28             @Override
29             public void onMessageReceived(ReceivedMessage message) {
30                 updateGraph(message);
31             }
32         };
33
34         setHasOptionsMenu(true);
35     }
36
37
38
39     @Override
40     public View onCreateView(LayoutInflater inflater, ViewGroup container,
41                              Bundle savedInstanceState) {
42         // Inflate the layout for this fragment
43         final View rootView = inflater.inflate(
44                         R.layout.fragment_detailed_graph, container, false);
45
46         graphView = rootView.findViewById(R.id.detailed_graphview);
47
```

```
48          graphView.addSeries(graph.getLineGraphSeries());
49          graphView.setTitle(graph.getGraphName());
50
51          if (graph.getMinY() != graph.getMaxY()) {
52              graphView.getViewport().setYAxisBoundsManual(true);
53              graphView.getViewport().setMinY(graph.getMinY());
54              graphView.getViewport().setMaxY(graph.getMaxY());
55              System.out.println("YAXISBOUNDS MANUAL TRUE");
56          }
57
58          if (graph.getMinX() != null || graph.getMaxX() != null) {
59              if (!graph.getMinX().equals(graph.getMaxX())) {
60                  graphView.getViewport().setXAxisBoundsManual(true);
61                  if (graph.getMinX() != null)
62                      graphView.getViewport().setMinX(graph.getMinX().
63                              getTime());
64                  if (graph.getMaxX() != null)
65                      graphView.getViewport().setMaxX(graph.getMaxX().
66                              getTime());
67                  System.out.println("XAXISBOUNDS MANUAL TRUE");
68              }
69          }
70
71
72          graphView.getGridLabelRenderer().setVerticalAxisTitle(graph.
73                  getyAxisDescription());
74          graphView.getGridLabelRenderer().setHorizontalAxisTitle("Date");
75
76          // styling grid/labels
77          graphView.getGridLabelRenderer().setHighlightZeroLines(true);
78          graphView.getGridLabelRenderer().
79                  setVerticalLabelsAlign(Paint.Align.LEFT);
80          graphView.getGridLabelRenderer().setLabelVerticalWidth(50);
81          graphView.getGridLabelRenderer().setTextSize(30);
82          graphView.getGridLabelRenderer().
83                  setGridStyle(GridLabelRenderer.GridStyle.HORIZONTAL);
84          graphView.getGridLabelRenderer().setHorizontalLabelsAngle(120);
85          graphView.getGridLabelRenderer().reloadStyles();
86
87          graph.getLineGraphSeries().setDrawDataPoints(true);
88          graph.getLineGraphSeries().setDataPointsRadius(10);
89          graph.getLineGraphSeries().setThickness(8);
90
91
92          // enable scrolling
93          graphView.getViewport().setScrollable(true);
94
95          // enable scaling
```

```
96          graphView.getViewport().setScalable(true);
97
98          // set date label formatter
99          final SimpleDateFormat dateFormat = new SimpleDateFormat();
100         dateFormat.applyPattern(GraphFragment.GRAPH_VIEW_PATTERN);
101         graphView.getGridLabelRenderer().setLabelFormatter(new
102                     DateAsXAxisLabelFormatter(
103                             graphView.getContext(), dateFormat));
104         graphView.getGridLabelRenderer().setNumHorizontalLabels(4);
105
106         graph.getLineGraphSeries().
107                     setOnDataPointTapListener(
108                     new OnDataPointTapListener() {
109             @Override
110             public void onTap(Series series, DataPointInterface dataPoint) {
111                 TextView textView = rootView.
112                             findViewById(R.id.detailed_graph_textview);
113                 dateFormat.applyPattern(
114                             GraphFragment.GRAPH_VIEW_EXTENDED_PATTERN);
115                 String date = dateFormat.format(
116                             new Date((long) dataPoint.getX()));
117                 textView.setText("X-Wert: " + date +
118                             System.getProperty("line.separator")
119                                     + "Y-Wert: " + dataPoint.getY());
120             }
121         });
122
123         return rootView;
124     }
125
126     public void updateGraph(ReceivedMessage message) {
127         if (graph.getGraphTopic().equals(message.getTopic())) {
128             PersistedMessage pMessage = PersistedMessage.
129                     convertToPersistedMessage(message);
130             double value = pMessage.getValueFromMessage();
131
132             graph.getLineGraphSeries().appendData(
133                     new DataPoint(pMessage.getTimestamp(), value),
134                             false, Graph.MAX_DATA_POINTS);
135         }
136     }
137
138 }
```

Listing 2: Auszug der Klasse 'DetailedGraphFragment'