JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

SYT Theorie

Sync bei mobilen Diensten

Systemtechnik Labor
5BHIT 2017/18, GruppeA

Nicolaus Rotter

Version 0.2
Begonnen am 18. April 2018
Beendet am 18. April 2018

Note:
Betreuer: Borko

Inhaltsverzeichnis

1 Einführung

Diese Übung soll die möglichen Synchronisationsmechanismen bei mobilen Applikationen aufzeigen.

1.1 Ziele

Das Ziel dieser Übung ist eine Anbindung einer mobilen Applikation an ein Webservices zur gleichzeitigen Bearbeitung von bereitgestellten Informationen.

1.2 Voraussetzungen

- Grundlagen einer höheren Programmiersprache
- Grundlagen über Synchronisation und Replikation
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von Webservices

1.3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die einen Informationsabgleich von verschiedenen Clients ermöglicht. Dabei ist ein synchronisierter Zugriff zu realisieren. Als Beispielimplementierung soll eine Ëinkaufsliste"gewählt werden. Dabei soll sichergestellt werden, dass die Information auch im Offline-Modus abgerufen werden kann, zum Beispiel durch eine lokale Client-Datenbank.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Wichtig ist dabei die Dokumentation der Vorgehensweise und des Designs. Es empfiehlt sich, die im Unterricht vorgestellten Methoden sowie Argumente (pros/cons) für das Design zu dokumentieren.

2 Vergleich Schnittstellen

2.1 Firebase

Firebase biete eine synchrone Plattform um Apps zu entwickeln. Der Grundgedanke hinter Firebase ist die Entwicklung mit häufigen live-Updates (also während der Laufzeit). Die Daten in Firebase Datenbank werden als JSON Files abgespeichert. Außerdem ist die Firebase DB in der CLoud und somit kann von überall ein Zugriff erfolgen.

Firebase bietet durch den Einsatz einer Zentralen *Realtime Database* die Funktion, dass sämtliche Clients sowie die Datenbank während der Laufzeit aktuell gehalten (synchronisiert) werden.

2.2 Parse Server

Parse Server ist die Open-Source Version von Parse und wurde von Facebook entwickelt. Die Infrastruktur von Parse Server läuft auf Node.js, nach dem Aufsetzen der Datenbank und dem Hinzufügen von Dateneinträgen kann über den Einsatz von *Express Web App* extrem simpel ein Client erstellt werden. (fast keine Code Änderungen notwendig)

So wie Parse nutzt Parse Server zum speichern von Daten MongoDB und zum speichern von File-Systemen Amazon S3 buckets.

Das Filesystem, in welchem die Daten letztendlich gespeichert werden kann vom User angegeben werden. Jedoch wird JSON empfohlen.

Das Dashboard von Parse Server biete dem User die Möglichkeit die Apps zu Verwalten und zu Konfigurieren, sowie Push Nachrichten zu senden.

Es können sogenannte ”live-Queries” erstellt werden, die bei der Abänderung von bestimmten Daten direkt eine Query ausführen. Somit erspart sich der User manuelle Querys.

2.3 DeltaDB

DeltaDB ist ähnlich zu Firebase weist jedoch ein paar Unterscheide auf:

- Geschrieben in JavaScript
- Client schreibt in eine lokale Offline DB und aktualisiert beim Verbinden mit dem Internet
- ist letzendlich Konsistent
- Extrem Skalierbar
- Hoch Verfügbar

3 Architektur

3.1 Client Server Architektur

Bei einer Verbidung über das Internet gibt es immer einen Host (Server), welcher Daten oder Dienste für den andern Host (Client) bereitstellt. Der Server reagiert und antwortet auf Anfragen während der Client diese an ihn stellt.

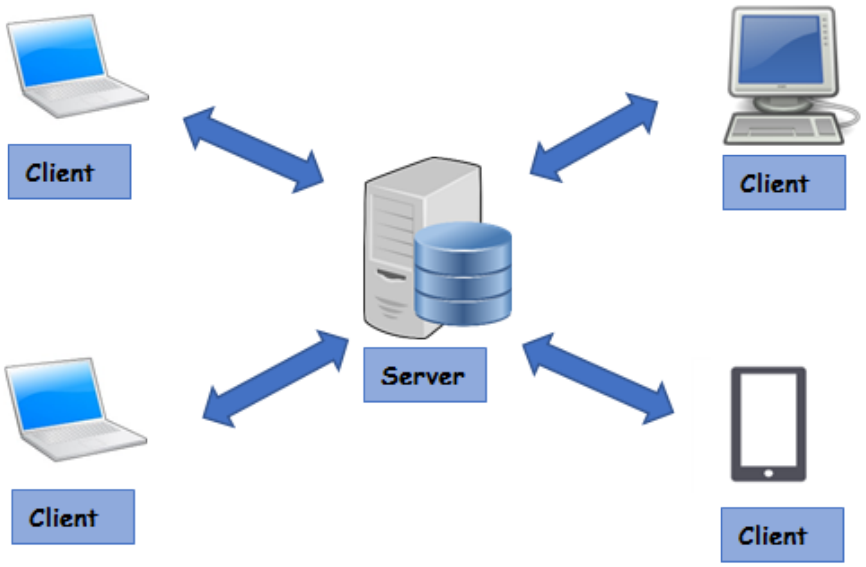

Abbildung 1: Clent-Serevr Architektur

Ein großer Vorteil dieser Architektur ist die Skalierbarkeit . Es können Problemlos weitere Clients, Server, oder andere Netzwerksegmente hinzugefügt werden. Nur 1 Server bedeutet auch, dass nur eine Sicherheitskomponente geschützt werden muss.

Ein Ausfall des Servers würde dann jedoch zum Ausfall des Services und des gesamten Netzes führen, da dieser die Zentral Komponente bildet.

4 Implementation

Zur Synchroniserung der Daten über sämtliche Clients wurde Firebase verwendet. Firebase ist eine NoSql Datenbank, synchronisiert sämtliche Daten auf allen Clients und macht diese auch Offline verfügbar.

Firebase selbst bietet 2 Datenbanken Varianten an, aus welchen ich mich für eine Real-Time Datenbank entschieden habe. Diese ist eine Cloud-basierte Datenbank, welche ihre Daten als JSON Files abspeichert.

4.1 Firebase Projekt erstellen

Zuerst muss also ein Firebase-Projekt angelegt werden. Dazu wird nach der Erfolgreichen Registrierung bei Firebase auf der Website auf "Create new Project" geklickt.

Anschließend kann man sich einen Namen für das Projekt aussuchen. Nachdem das Projekt erstellt wurde kann über die "Add Firebase to your Android App" und der Angabe des Package Namen (bei mir firebase.com.shoppinglistapp), welcher im Projekt verwendet wird ein google-services.json File heruntergeladen werden.

4.2 Adroid App Projekt erstellen

Über Adroid Studio wird nun ein neues Projekt mit dem Namen ShoppingListApp erstellt. Dieses wird später über die *API 16: Adroid 4.1(Jelly Bean)* Version gestartet, anschließend wird eine *Empty Activity* erstellt und das Projektsetup beendet.

4.3 Firebase mit Projekt verbinden

Nun nimmt man das zuvor gedownloadete *google-services.json* File und legt es in der root Path des Projekts (ShoppingListApp/app/). Weitere Schritte bezüglich gradle dependencys sind auf der Firebase Website zu finden.

Es werden also dem build.Gradle auf Projektebene folgende Dependency hinzugefügt:

```
1 classpath 'com.google.gms:google-services:3.0.0'
```

Und dem build.Gradle auf App Ebene

```
1 compile 'com.google.firebase:firebase-database:9.0.2'
```

Und am Ende des Files:

```
1 apply plugin: 'com.google.gms.google-services'
```


4.4 Firebase auf Testverion setzen

Damit die App nun ohne Authentifizierung Zugriff auf die Firebase Datenbank hat, wird in der Firebase Console im Reiter *Database* die Option "Im Testmodus starten" ausgewählt.

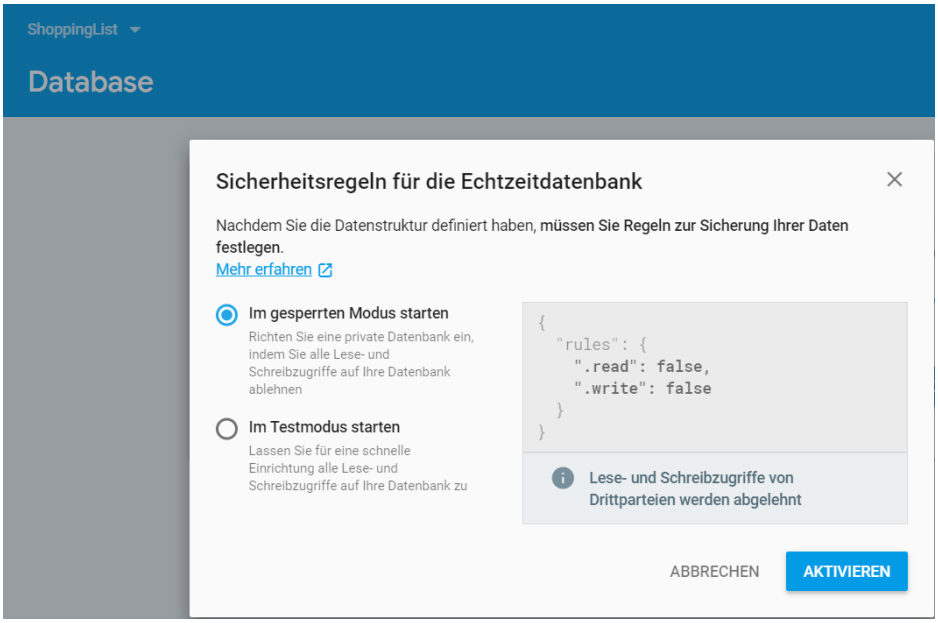

Abbildung 2: Rules of RealtimeDB

4.5 build.Gradle

Nun werden die Projekt build.Gradle dependencys um folgende erweitert:

```
classpath 'com.android.tools.build:gradle:3.0.1'
classpath 'com.google.gms:google-services:3.2.0'
classpath 'com.google.gms:google-services:3.0.0'
```

Das App build.Gradle File wird um folgendes erweitert:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.android.support:design:26.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.google.firebase:firebase-database:9.0.2'
}
apply plugin: 'com.google.gms.google-services'
```

4.6 Internetzugriff

Um der App Iternetzugriff zu gewähren wird im Manifest.xml File noch folgende Zeile hinzugefügt:

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

4.7 Strings.xml

Außerdem muss im Strings.xml File (res/values/Strings.xml) folgende Ressource hinzugefügt werden um später Zugriff zu haben:

```
1 <resources>
      <string name="app_name">ShoppingListApp</string>
3     <string name="action_settings">Settings</string>
      <string name="add_task_button">Add Task</string>
5 </resources>
```

4.8 Main Activity

4.8.1 Main Activity Layout

Nun wird das Main Activity Layout File erstellt, dazu kann im Design Tab der Android Studios eine UI erstellt werden. Diese soll folgende Elemente aufweisen:

- RecyclerView: Die View, in welcher später die ShoppingList Items angezeigt werden
- EditText: Ein Input-Feld um den Namen eines neuen Items einzugeben
- Button: Add Item Button um das Item hinzuzufügen

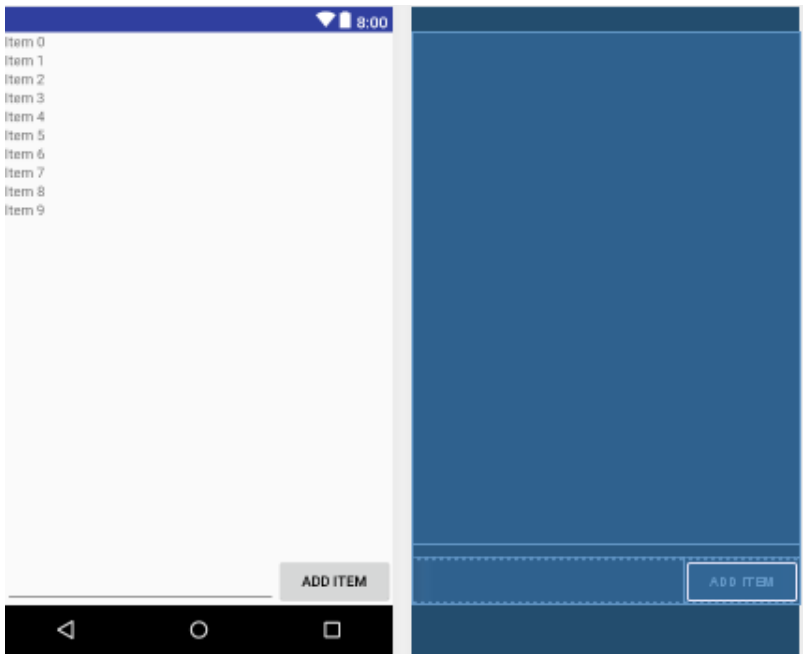

Abbildung 3: Main Application UI

4.8.2 Main Activity Java

Die zugehörige Java Klasse *MainActivity.java* bildet die Funktinonen der UI.

Beim klicken des Buttons wird der Inhalt des EditText Feldes gelesen und über die Push() und setValue() Methoden an die Datenbank gesendet. AUßerdem werden Items mit weniger als 6 Zeichen und Leere Items über eine Errormeldung abgefangen.

Hier der Code hinter der onClick() Methode desButtons:

```
addTaskButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String enteredTask = addTaskBox.getText().toString();
        if(TextUtils.isEmpty(enteredTask)){
            Toast.makeText(MainActivity.this, "You must enter a task first", Toast.LENGTH_LONG).show();
        return;
    }
    if(enteredTask.length() < 6){
        Toast.makeText(MainActivity.this, "Task count must be more than 6", Toast.LENGTH_LONG).show();
        return;
    }
    Task taskObject = new Task(enteredTask);
    databaseReference.push().setValue(taskObject);
    addTaskBox.setText("");
    }
});
```

Danach wird ein ChildListener() Implementiert, welcher auf Veränderungen der Daten in der Firebase Datenbank reagiert. Falls also ein Item hinzugefügt wird oder verändert, wird durch die Methode *getAllTask()* die Liste der Items aktualisiert. Falls ein Child entfernt wird wird über die Methode *taskDeletion()* dieser entfernt.

```
databaseReference.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        getAllTask(dataSnapshot);
    }
    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String s) {
        getAllTask(dataSnapshot);
    }
    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
        taskDeletion(dataSnapshot);
    }
});
```

Hier noch der Code der beiden Methoden *getAllTask*:

```
private void getAllTask(DataSnapshot dataSnapshot){
    for(DataSnapshot singleSnapshot : dataSnapshot.getChildren()){
        String taskTitle = singleSnapshot.getValue(String.class);
        allTask.add(new Task(taskTitle));
        recyclerViewAdapter = new RecyclerViewAdapter(MainActivity.this, allTask);
        recyclerView.setAdapter(recyclerViewAdapter);
    }
}
```

und *taskDeletion*:

```
private void taskDeletion(DataSnapshot dataSnapshot){
    for(DataSnapshot singleSnapshot : dataSnapshot.getChildren()) {
        String taskTitle = singleSnapshot.getValue(String.class);
        for(int i = 0; i < allTask.size(); i++){
            if(allTask.get(i).getTask().equals(taskTitle)){
            allTask.remove(i);
        }
    }
    Log.d(TAG, "Item tile " + taskTitle);
    recyclerViewAdapter.notifyDataSetChanged();
    recyclerViewAdapter = new RecyclerViewAdapter(MainActivity.this, allTask);
    recyclerView.setAdapter(recyclerViewAdapter);
}
```

4.9 RecyclerView

4.9.1 RecyclerViewHolders.java

Die RecyclerViewHolder Klasse fügt den Delete-Icons des einen Listener hinzu, welchen beim Drücken des Icons einen Toast (Flashmeldung) ausgibt und anschließend das Item, welches mit dem Icon verbunden ist aus der liste löscht.

```
deleteIcon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(v.getContext(), "Delete icon has been clicked", Toast.LENGTH_LONG).show();
        String taskTitle = taskObject.get(getAdapterPosition()).getTask();
        Log.d(TAG, "Task Title " + taskTitle);
        DatabaseReference ref = FirebaseDatabase.getInstance().getReference();
        Query applesQuery = ref.orderByChild("task").equalTo(taskTitle);
        applesQuery.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (DataSnapshot appleSnapshot: dataSnapshot.getChildren()) {
                    appleSnapshot.getRef().removeValue();
                }
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {
                Log.e(TAG, "onCancelled", databaseError.toException());
            }
        });
    }
});
```


4.10 List View

Anschließend wird eine View für die Item Liste erstellt. Diese enthält zuerst eine IMageView für das Shoppingcart Icon, eine TextView für den Namen des Items und eine ImageView für das Delete Icon zum Löschen des Items-Eintrags.

Diese sieht folgendermaßen aus:

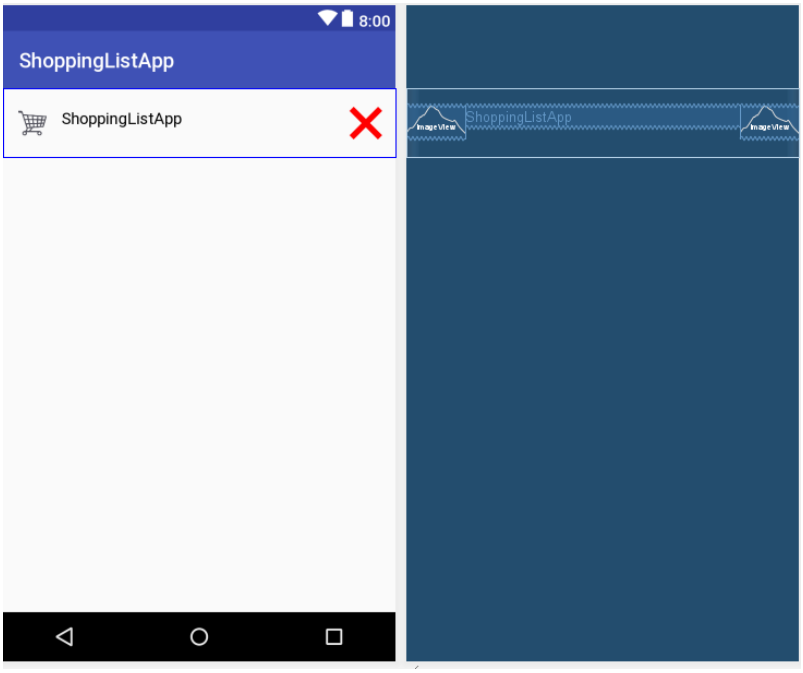

Abbildung 4: ListView

xml-Code dazu:

```
<ImageView
android:id="@+id/task_icon"
android:layout_width="0dp"
android:layout_height="32dp"
android:src="@mipmap/shoppingcart_front"
android:layout_weight="15"
android:contentDescription="@string/app_name" />
<TextView
android:id="@+id/task_title"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="70"
android:text="@string/app_name"
android:textSize="16sp"
android:textColor="@color/colorBlack"/>

<ImageView
android:id="@+id/task_delete"
android:layout_width="0dp"
android:layout_height="32dp"
android:layout_weight="15"
android:contentDescription="@string/app_name"
android:src="@drawable/redx" />
```

4.10.1 Custom Icons

Um hier ein eigenes Icon einzufügen, muss man per Rechtsklick auf den "res" Ordner dann "New" -> "Image Asset" ein neues Asset hinzufügen. Ich habe hier ein Rotes X und ein Shoppingcart erstellt:

(a) Rotes X Icon (b) Shoppingcart Icon

Abbildung 5: Verwendete Custom Icons

Daher auch der Aufruf per:

```
adroid:src="@mipmap/shoppingcart_front"
android:src="@mipmap/redx_front"
```

4.11 Task

Zuletzt wird noch eine Klasse für Task Objekte (Items) erstellt.

```
public class Task {
    private String task;
    public Task() {}
    public Task(String task) {
        this.task = task;
    }
    public String getTask() {
        return task;
    }
}
```

5 Ausführen

5.1 Emulator

Somit ist die Implemetation fertig und die fertige App kann über den Vitual Device Viewer gestartet und getestet werden. Diese sieht dann nach 3 Einträgen folgendermaßen aus:

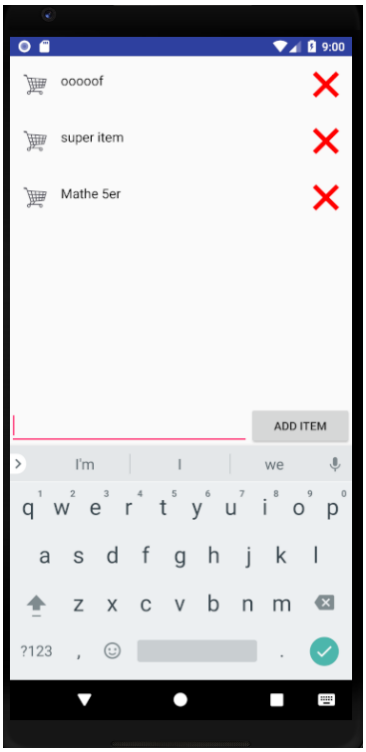

Abbildung 6: Emulator Screenshot

5.2 Firebase Konsole

Nebenbei kann man in der Firebase Konsole die Änderungen am JSON File (also der Datenbank) mitverfolgen, hier wird pro Eintrag ein Key erstellt und unter diesem der in der TextEdit eingegeben Name für ein Item:

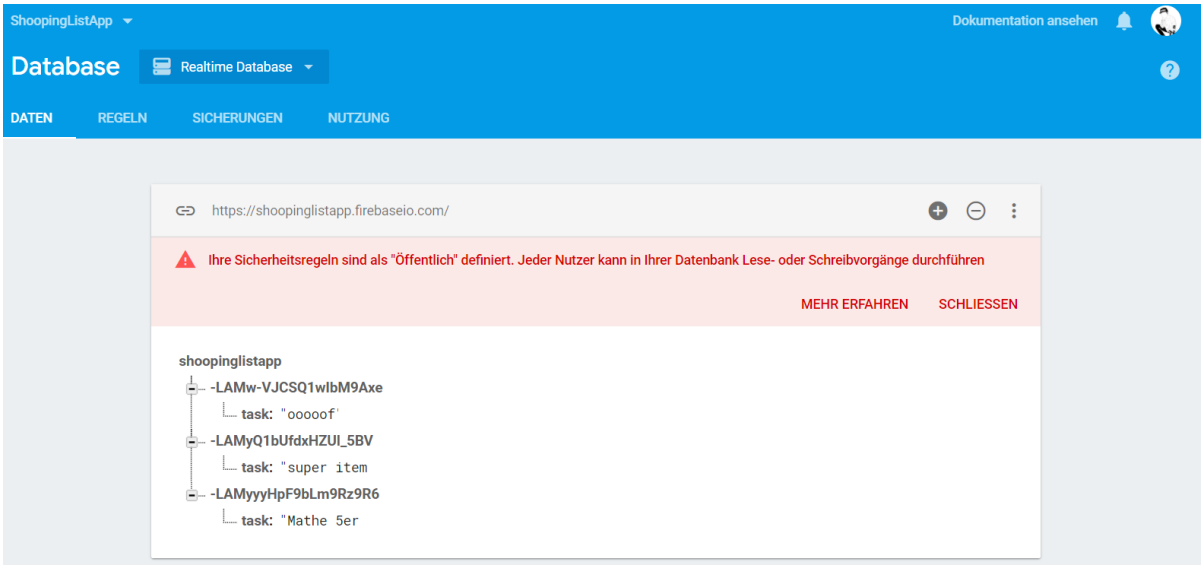

Abbildung 7: Firebase Console

5.3 APK

Zuletzt kann man über Android Studio durch das Klicken von ”Build” -> ”Build APKs” eine APK der App erstellen und diese anschließend auf ein Android Handy laden. Ich habe diese auf mein Handy geladen und zusammen mit dem Emulator die Synchronität der Daten in der Liste überprüft.

Wenn also am Handy ein Eintrag gelöscht wird, wird dieser auch im Emulator und in der Datenbank entfernt.

Literatur

[1] A.S. Tanenbaum and M. Van Steen. *Verteilte Systeme: Prinzipien und Paradigmen.* Pearson Studium. Addison Wesley Verlag, 2007.

Tabellenverzeichnis

Listings

Abbildungsverzeichnis