JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Synchronisation bei mobilen Diensten

Systemtechnik
5BHIT 2017/18

Matthias Weiss

Version 1
Note:
Begonnen am 17. April 2018
Betreuer: Michael Borko
Beendet am 18. April 2018

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.

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen „Grundkompetenz überwiegend erfüllt“
  - Beschreibung des Synchronisationsansatzes und Design der gewählten Architektur (Interaktion, Datenhaltung)
  - Recherche möglicher Systeme bzw. Frameworks zur Synchronisation und Replikation der Daten
  - Dokumentation der gewählten Schnittstellen
- Anforderungen „Grundkompetenz zur Gänze erfüllt“
  - Implementierung der gewählten Umgebung auf lokalem System
  - Überprüfung der funktionalen Anforderungen zur Erstellung und Synchronisation der Datensätze
- Anforderungen „Erweiterte-Kompetenz überwiegend erfüllt“
  - CRUD Implementierung

  - Implementierung eines Replikationsansatzes zur Konsistenzwahrung

- Anforderungen „Erweiterte-Kompetenz zur Gänze erfüllt“
  - Offline-Verfügbarkeit
  - System global erreichbar

2 Abgabe

Abgabe bitte den Github-Link zur Implementierung und Dokumentation (README.md).

3 Ergebnisse

3.1 Möglichkeiten zur Datenhaltung

3.1.1 Websocket Implementation

Es ist möglich eigene Server Nodes zu entwicklen, welche die Clients mithilfe von Websockets verwalten.

3.1.2 Couchbase Mobile

Couchbase Mobile ist eine Zusammensetzung aus Software, die eine NoSQL Datenbank darstellt und für Android und IOS Applikationen ausgelegt ist. Dies und die Synchronisation mit dem Couchbase Server und anderen Endgeräten erfolgt mit der Verwendung von Couchbase Lite und Couchbase Sync Gateway.

3.1.3 Firebase

Firebase ist eine Platform, welche von Google entwickelt wurde und viele Features für die Entwicklung von mobilen Applikationen bietet. Hauptsächlich auf Android und IOS. Außerdem bietet es Features zur Überwachung und Analyse der Datenbanken.

4 Umsetzung

Als Hilfe bei der Umsetzung der mobilen Applikation wurde ein Tutorial hergenommen. Da mit diesem aber keine Einkaufsliste erstellt wird, musste diese Applikation noch etwas abgeändert werden.
http://www.appsdeveloperblog.com/todo-list-app-kotlin-firebase/

4.1 Verwendete Programmiersprache

Bei der Auswahl der Programmiersprache wurde sich an das Tutorial gehalten. Es wurde die Programmiersprache Kotlin verwendet.
Kotlin ist eine statisch typisierte Programmiersprache, welche für die JVM (Java Virtual Machine) übersetzt wird. Der Bytecode kann auch in JavaScript Quellcode umgewandelt werden. Kotlin wird von dem JetBrains Programmierern weiterentwickelt.

4.2 Verwendete Datenhaltungsmöglichkeit

Entschieden wurde sich für Firebase, da es eine der „stärksten“ Echtzeit Datenbanken-API ist. Außerdem ist die Anbindung an eine Firebase Datenbank in Android Studio um einiges unumständlicher als bei z.B.: Couchbase.

4.3 Projekt aufsetzen

Entwickelt wurde mit der Entwicklungsumgebung **Android-Studio**. Wenn Android Studio gestartet wurde, wird zuerst ein neues Projekt erstellt. Hierbei ist wichtig, dass ein Häkchen bei **Include Kotlin support** gesetzt wird bevor das Projekt erstellt wird.

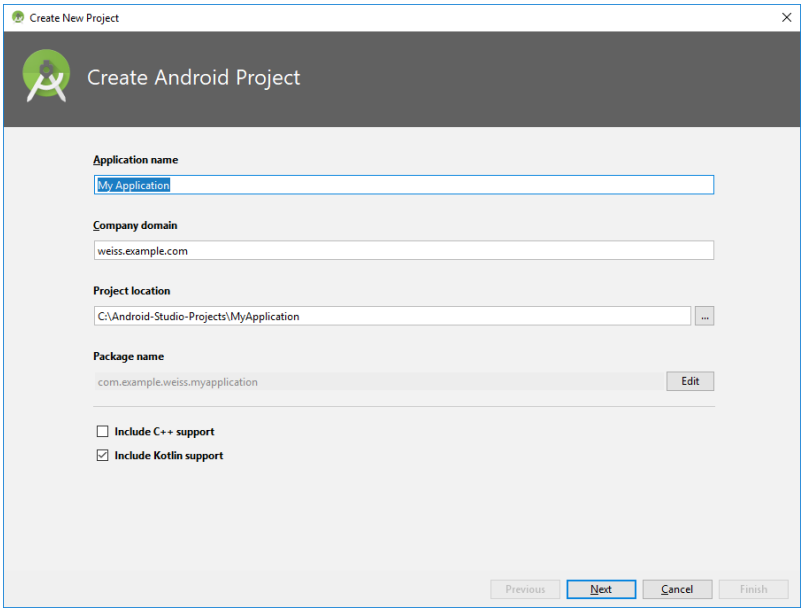

Abbildung 1: Projekt erstellen

Wenn das Projekt erstellt wurde, wird als nächstes die Firebase Unterstützung hinzugefügt. Hierzu muss oben in der **Toolbar** auf **Tools** > **Firebase** geklickt werden.

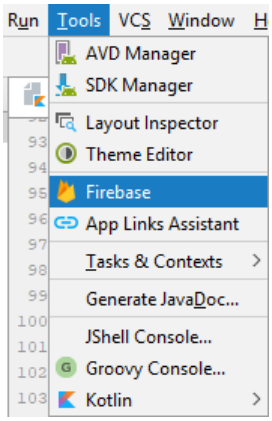

Abbildung 2: Firebase Support

Dann erscheint auf der rechten Seite in Android Studio der Assisten für Firebase. Hier muss sich zur Datenbank verbunden werden, indem auf **Realtime Database** geklickt wird. Dort wird zur Datenbank verbunden und der Applikation kann die Datenbankverbindung hinzugefügt werden.

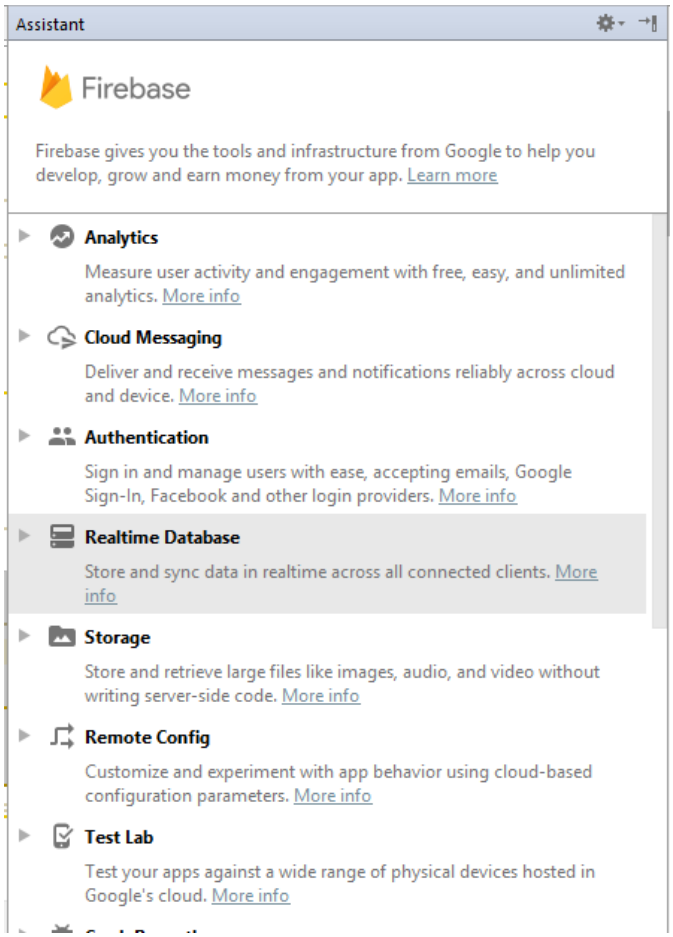

Abbildung 3: Firebase Assistent

Wenn dies fertig ist, wird als nächstes auf der |Firebase Konsole| im Browser unter dem Reiter **Develop** > **Database** eine Realtime Datenbank hinzugefügt und im **Testmodus** gestartet. Dieser Modus ermöglicht es jedem der die ID dieser Datenbank hat darauf zuzugreifen.

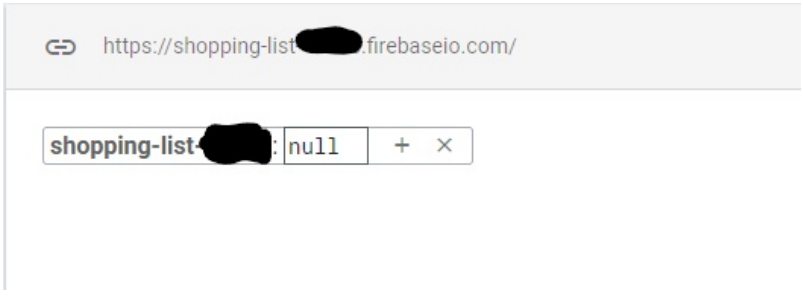

Abbildung 4: Datenbank im Browser

Nun kann mit der Implementierung begonnen werden. Zuerst wird eine Kotlin Datei erstellt, damit der Firebase Datenbank eine Collection hinzugefügt wird. Diese wird shop-item genannt.

```
object Constants {
    @JvmStatic val FIREBASE_ITEM: String = "shop_item"
}
```

In der selben Datei wird auch noch ein Model für ein **ShopItem** erstellt. Dies ist eine Klasse in der eine Objekt-ID, ein Text als Beschreibung und eine Boolean Variable, welche verwendet wird um zu überprüfen ob dieses Item angehakelt wurde.

```
class ShopItem {
    companion object Factory {
        fun create(): ShopItem = ShopItem()
    }
    var objectId: String? = null
    var itemText: String? = null
    var done: Boolean? = false
}
```

Dann wird die App designed. Dies ist jedem selbst überlassen wie diese aussieht. Funktionalitäten, welche der App hinzugefügt wurden.

- ein |FloatingButton|, welcher ein Dialog Fenster öffnet um Items hinzuzufügen
- eine **ListView** und eine **Checkbox** in jeder Reihe
- wenn die Checkbox geklickt wird, dann wird diese als „Fertig“ angezeigt
- ebenfalls in jeder Reihe ein **DeleteButton**, um die Einträge wieder heraus zu löschen

Diese sieht dann ungefähr wie folgt aus:

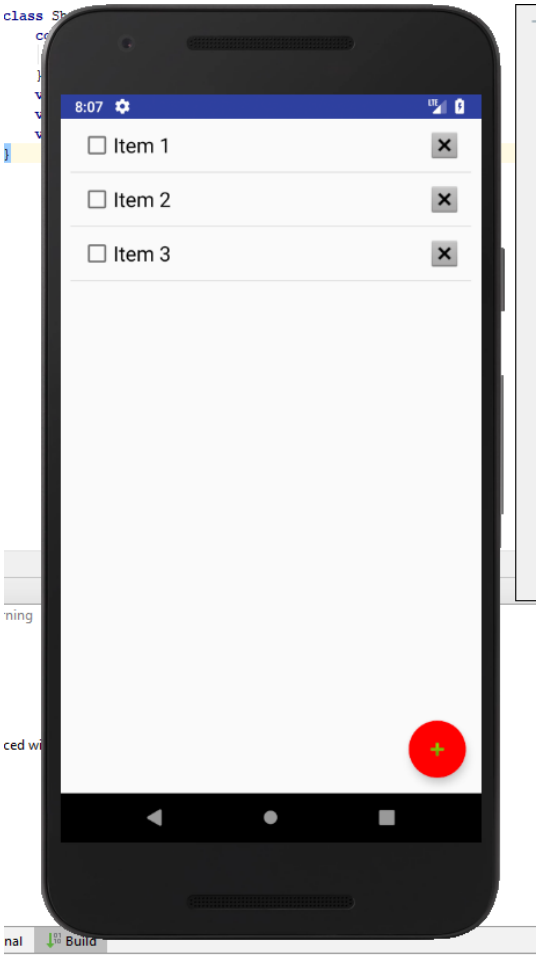

Abbildung 5: Design der App

4.4 Dialog Fenster bei Button-Klick

Diese Methode erstellt ein Dialog Fenster, in dem ein neues Item hinzugefügt werden kann. Weiters wird dem **Submit Button** die Funktion gegeben die neu eingegebenen Daten in die Firebase Datenbank zu speichern, dies geschieht mit **.push()**.

```
private fun addNewItemDialog() {
    val alert = AlertDialog.Builder(this)
    val itemEditText = EditText(this)
    alert.setMessage("Add New Item")
    alert.setTitle("Enter To Do Item Text")
    alert.setView(itemEditText)
    alert.setPositiveButton("Submit") { dialog, positiveButton ->
        val shopItem = ShopItem.create()
        shopItem.itemText = itemEditText.text.toString()
        shopItem.done = false
        val newItem = mDatabase.child(Constants.FIREBASE_ITEM).push()
        shopItem.objectId = newItem.key
        newItem.setValue(shopItem)
        dialog.dismiss()
        Toast.makeText(this, "Item saved with ID " + shopItem.objectId, Toast.LENGTH_SHORT).show()
```

```
16      }
        alert.show()
18  }
```

Wenn man die Datenbank im Browser aufruft, werden jetzt alle hinzugefügten Daten angezeigt.

Abbildung 6: Daten in der Datenbank

4.5 Daten in Liste einfügen

Die folgende Methode wird dafür verwendet um in der App alle Einträge in einer Liste darzustellen. Dafür werden alle Daten aus der Datenbank ausgelesen und nacheinander in die Liste der App eingefügt. Hier muss aber vor dem Einfügen aller Daten die Liste in der App nochmals „gecleared“ (**.clear()**) werden, ansonsten würden alle Daten öfters in die Liste eingetragen werden.

```
    private fun addDataToList(dataSnapshot: DataSnapshot) {
2       val items = dataSnapshot.children.iterator()
        shopList!!.clear();
4       if (items.hasNext()) {
            val shopListindex = items.next()
6           val itemsIterator = shopListindex.children.iterator()

8           while (itemsIterator.hasNext()) {
                val currentItem = itemsIterator.next()
10              val shopItem = ShopItem.create()
                val map = currentItem.getValue() as HashMap<String, Any>
12              shopItem.objectId = currentItem.key
                shopItem.done = map.get("done") as Boolean?
14              shopItem.itemText = map.get("itemText") as String?
                shopList!!.add(shopItem);
16          }
        }
18      adapter.notifyDataSetChanged()
    }
```

4.6 Daten checken

Mit einem Klick auf die Checkbox wird diese „aktiviert“ und der Boolean wert in der Datenbank wird verändert, damit auf allen Geräten die Checkbox geändert wird.

```
override fun modifyItemState(itemObjectId: String, isDone: Boolean) {
    val itemReference = mDatabase.child(Constants.FIREBASE_ITEM).child(itemObjectId)
    itemReference.child("done").setValue(isDone);
}
```

4.7 Daten löschen

Ein Klick auf den **Löschen Button** löscht den Eintrag aus der Liste und aus der Datenbank, damit auch auf allen anderen Geräten dieser Eintrag nicht mehr angezeigt wird.

```
override fun onItemDelete(itemObjectId: String) {
    val itemReference = mDatabase.child(Constants.FIREBASE_ITEM).child(itemObjectId)
    itemReference.removeValue()
}
```

4.8 Firebase Offline Persisting

Firebase kann Daten sehr einfach und schnell **cachen**. Firebase wird die Daten der Datenbank auf deinem Gerät zu cachen und wenn die App wieder Online geht, dann werden die Daten mit der Datenbank abgeglichen und aktualisiert.
Hierzu wird eine Application Klasse erstellt in der diese Funktionalität aktiviert wird.

```
class ThisApplication: Application() {
    override fun onCreate() {
        super.onCreate()
        FirebaseDatabase.getInstance().setPersistenceEnabled(true)
    }
}
```

Listings

Abbildungsverzeichnis