JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Synchronisation mobile Dienste

Systemtechnik Theorie
5BHIT 2017/18

Martin Wölfer

Version 0.1
Begonnen am 16. April 2018
Beendet am 17. April 2018

Note:
Betreuer: BORM

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

1.5 Abgabe

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

2 Ergebnisse

Es wurde entschieden eine Applikation mittels `React Native` und `Firebase` zu schreiben. Für die Datenspeicherung, Datenverwaltung und Datensynchronisation wurde die Cloud Datenbank `Cloud Firestore` verwendet.

2.1 Anforderungen

Es waren bestimmte Anforderungen gegeben, welche von der verwendeten Architektur bzw. Technologie erfüllt werden mussten:

- Daten speichern
- Daten lesen
- Daten ändern
- Daten löschen
- Daten synchronisieren
- Öffentlich ansprechbar

2.2 Firebase

Firebase ist eine Plattform entwickelt von Google welche eine Vielzahl von Features anbietet, hauptsächlich für mobile Applikation, namentlich Android und iOS. Firebase bietet eine Grundlage für viele technische Anforderungen und macht es auch möglich nach dem Release auf Nutzer einzugehen mittels diversen Analysetools, gezielten Werbungen, etc.. [1]

2.2.1 Cloud Firestore

Cloude Firestore ist eine der vielen Features welche angeboten werden. Cloud Firestore ist eine flexible, skalierbare NoSQL Datenbank welche in der cloud agiert und es ermöglicht Daten zwischen Server und Client zu synchronisieren.[2]

Hierbei wurde zwischen Cloud Firestore und Realtime Database entschieden. Beide Produkte lösen das Problem der echtzeitigen Datensynchronisierung. Der große Unterschied ist jedoch wie Daten gespeichert werden. Während Realtime Database die Daten in einem großen JSON Objekt speichert, werden in Cloud Firestore Daten in Dokumenten abgespeichert. Dadurch ist es leichter Daten in besonders großen Projekten zu organisieren und es ist leichter zu skalieren.

Weiters ist es mit Cloud Firestore möglich die Applikation als Webservice anzubieten und es werden indexierte Queries verwendet um die Performance zu erhöhen.

Der große Nachteil and Cloud Firestore ist, dass das Produkt noch immer in der Beta-Phase ist. Somit ist Realtime Database stabiler und verlässlicher. Da dieses kleine Demoprojekt allerdings nicht 100% zuverlässig sein muss, wurde Cloud Firestore verwendet um sich mit der Zukunft von Cloud-Datenspeicherung auseinanderzusetzen. [3]

2.2.2 Datenmodell

Wie bereits erwähnt ist Cloud Firestore keine herkömmliche SQL Datenbank mit Spalten und Reihen. Stattdessen werden die Daten in `documents` abgespeichert. Diese `documents` werden weiterführend in `collections` organisiert.

Jedes `document` enthält eine beliebige Anzahl an `key-value` Paaren. Es ist vergleichbar mit dem Aufbau und der Anwendung von JSON-Objekten.[4]

Documents Ist die „Einheit“ der Speicherung in Cloud Firestore. Ein Dokument welches ein Shopping-Artikel enthält ist folgendermaßen aufgebaut:

```
bought : false
title : "Cookies"
```

Es ist allerdings, wie in JSON, auch möglich verschachtelte Attribute zu definieren:

```
bought : false
title :
    product: "Cookies"
    company : "Milka"
```

[4, Abs. Documents]

Collections Sind als „Container“ für die einzelnen `documents` zu verstehen. Somit ist es leichter diese zu organisieren und zu verwalten. Im Beispiel der Einkaufliste, werden die einzelnen Artikel in Einkauflisten organisiert, wobei eine Einkaufsliste eine `collection` darstellt. [4, Abs. Collections]

2.2.3 Synchronisation

Um Echtzeit-Synchronisation zu ermöglichen wird eine Funktion bereitgestellt welche aufgerufen wird sobald eine Änderung in der Datenbank, `collection` oder `document` auftritt, `onSnapshot()`. Hierbei ist wichtig das entschieden kann auf was genau gehört werden soll. Wenn nur eine bestimmte `collection` oder sogar nur ein `document` von Interesse ist, kann mit `onSnapshot()` nur auf Änderungen dieser Objekte gehorcht werden. Beim ersten Aufruf wird ein gesamter Snapshot des angefragten Objektes mitgeliefert, bei weiteren Änderungen werden nur mehr die Änderungen mitgeliefert um die Performance zu verbessern.

Durch folgenden Code kann auf Änderungen der Einkaufliste gehorcht werden:

```
// Referenz in einer Variable speichern
this.ref = firebase.firestore().collection('shoppinglist');
// onSnapshot liefert automatisch eine Funktion zurueck welche aufgerufen werden kann um sich von
    Updates "abzumelden"
this.unsubscribe = this.ref.onSnapshot(this.onCollectionUpdate.bind(this))
```

2.2.4 Schnittstellen

Die wichtigsten Schnittstellen in Cloud Firestore sind folgende[5]:

- **app** (firebase.firestore.app)

  Ist die Applikation welcher mit dieser Firestore Instanz assoziiert ist

- **batch()** (firebase.firestore.WriteBatch)

  Mit dieser Funktion können mehrere Einträge auf einmal in die Datenbank geschrieben werden

- **collection()** (firebase.firestore.CollectionReference)

  Diese Funktion liefert die Referenz der als Parameter angegeben `collection` zurück. Ist die wahrscheinlich wichtigste Funktion, da mit der Referenz Dokumente geschrieben, gelöscht und verwaltet werden können.

- **doc()** (firebase.firestore.DocumentReference)

  Diese Funktion liefert die Referenz des als Parameter angegebenen `documents` zurück. Somit können nur einzelne Dokumente bearbeitet werden.

2.3 Alternative Technologien

- **Couchbase Mobile**

  Ist ebenfalls eine NoSQL Datenbank welche es ermöglicht Daten zwischen mobilen Applikation auszutauschen. [6]

- **Deployd**

  Verspricht eine Open-Source Alternative zu Firebase darzustellen [7]. Deployd hat allerdings nicht annähernd das breitgefächerte Arsenal an Features und kaum Community support.

- **Firehose**

  Ist ein minimalistischer Ansatz um Clients und Server synchron zu halten wie Websockets. [8]

- **„Händisch“ mit Websockets**

  Natürlich ist es auch möglich selber einen Node Server aufzusetzen und auf diesen alle Clients mit Websockets zu verwalten.

2.4 Umsetzung

2.4.1 Projekt aufsetzen

Um das Projekt aufzusetzen wurde das Projekt react-native-firebaser-starter als Basis verwendet. Dieses enthält zusätzlich ein Tutorial [9] welches verwendet wurde um die Applikation korrekt aufzusetzen.

Der erste Schritt war es im Basisprojekt alle nötigen Dependencies (`node_modules/`) zu installieren mittels `npm install`. Der nächste Schritt war es das Projekt einen Namen zu geben. Dafür gibt es ein Script welches mittels `npm run rename` ausgeführt wird, anschließend wird nach Name und Organisation gefragt und man erhält folgende Ausgabe:

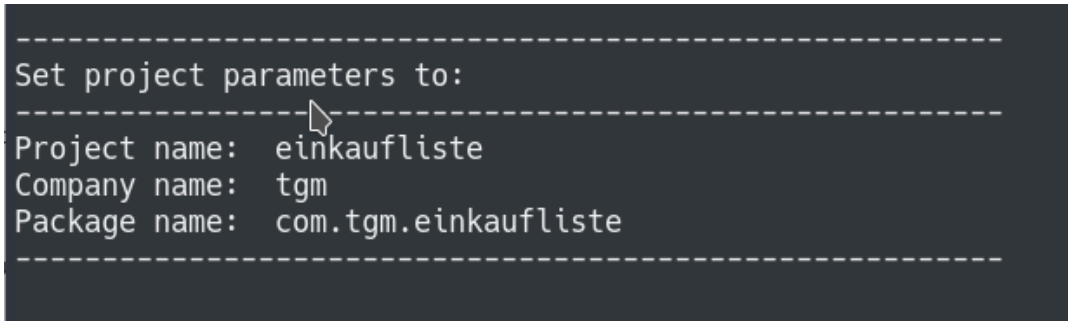

Abbildung 1: Projektparameter wurden angegeben

2.4.2 Firebase Projekt anlegen

Der nächste Schritt war es mit den angelegten Daten ein Firebase Projekt anzulegen und anschließend auszuwählen dass Firebase zu der Applikation hinzugefügt werden soll. Hierbei muss der Packagename angegeben werden, welcher im vorigen Schritt durch Angabe von Name und Organisation definiert wurde. Zusätzlich ist noch möglich ein Nickname und ein Signaturzertifikat anzugeben.

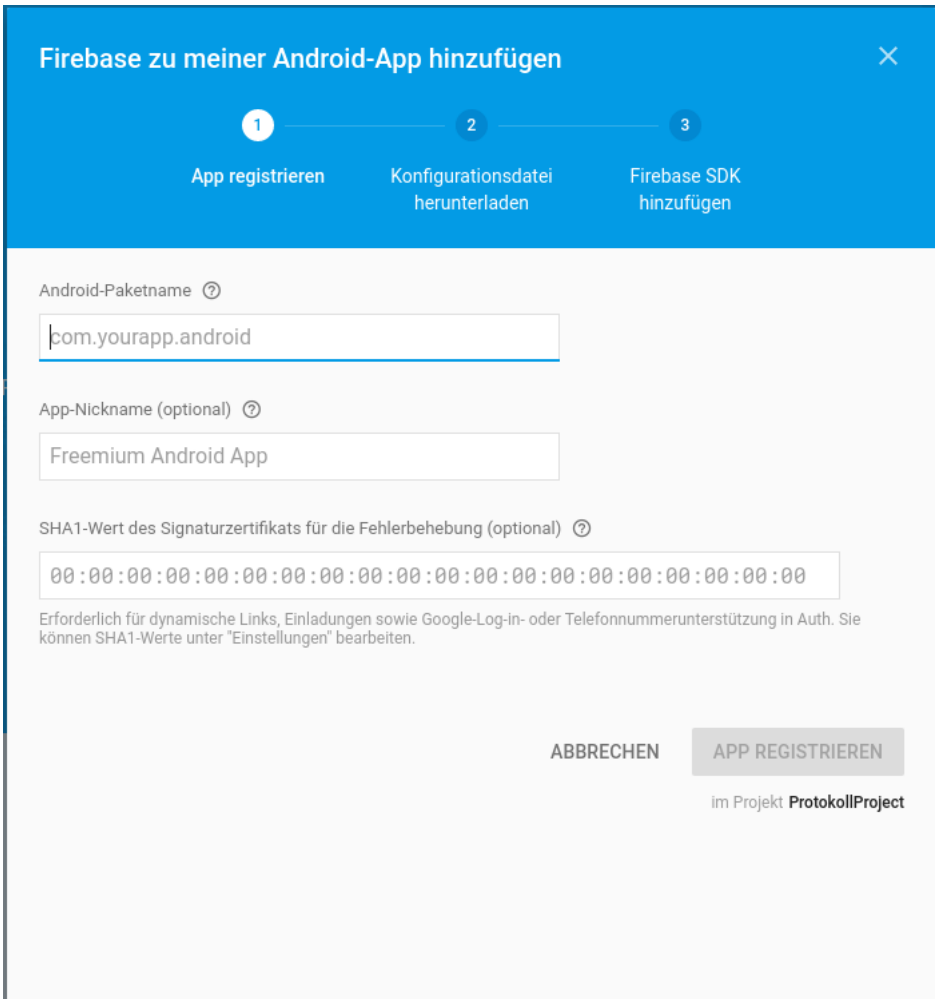

Abbildung 2: Firebase dem Projekt hinzufügen

Im nächsten Schritt kann nun ein Konfigurationsfile heruntergeladen werden, `google-services.json`, dieses muss anschließend im Ordner `android/app/` gespeichert werden.

Falls man alles richtig gemacht hat sollte es nun möglich sein die Demoapplikation auszuführen mit `react-native start` und `react-native run-android`. Man sollte mit einem Logo, Text und allen aktivierten Firebase Modulen begrüßt werden.

2.4.3 Daten einfügen

Wie bereits beschrieben, wurde zuerst eine Referenz auf die `shoppinglist`-Collection erstellt. Mit dieser Referenz können Daten hinzugefügt werden. Beim drücken des Submit-Buttons, wird die Funktion `addItem()` aufgerufen, welche mittels der Referenz und dem Text welcher eingegeben wurde ein neues Dokument anlegt:

```
this.ref.add({
    title: this.state.textInput,
    bought: false,
});
```

Nach Hinzufügen eines Artikels kann nun auf der Firebase Konsole [10] eingesehen werden, ob tatsächlich in der Collection ein neues Dokument angelegt wurde:

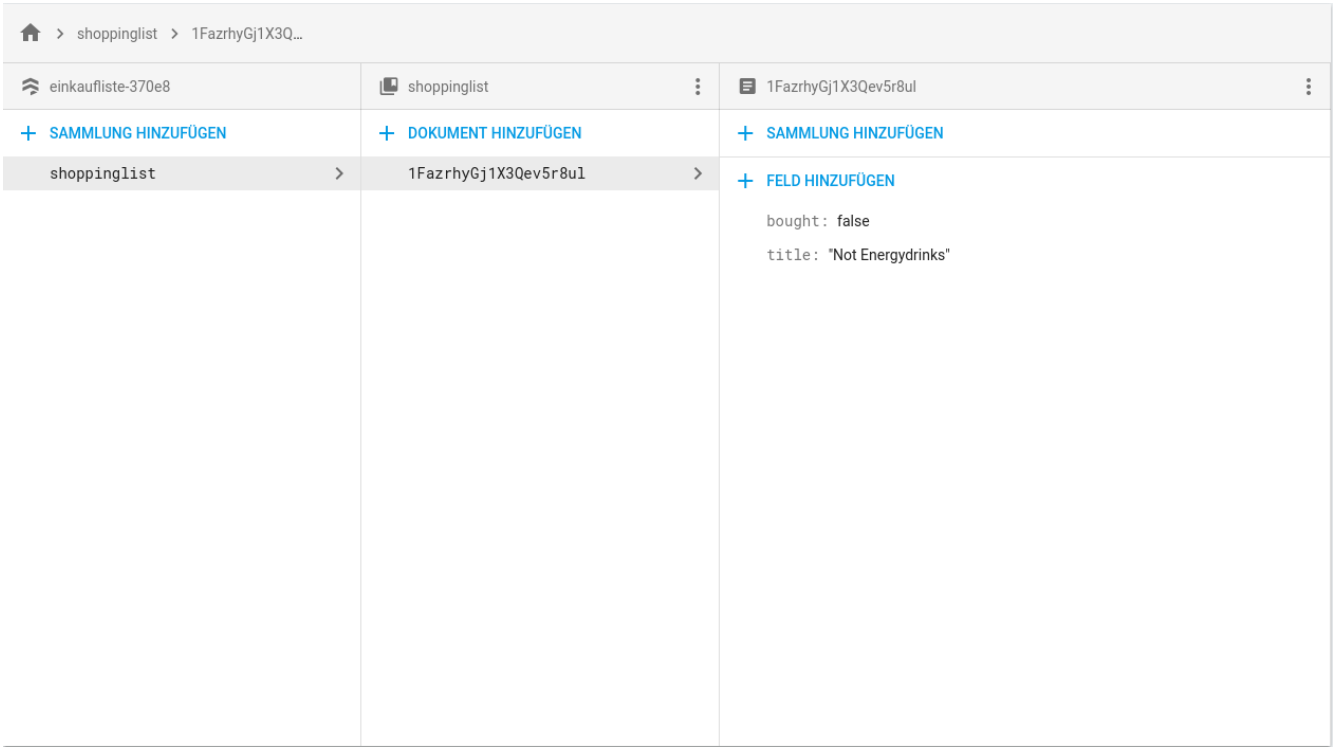

Abbildung 3: Der Collection wurde ein neues Dokument hinzugefügt

2.4.4 Daten anzeigen

Die Daten werden in einer `FlatList` angezeigt. Mit der Flatlist ist es möglich, ein Array an Daten anzugeben welches angezeigt werden soll, und zu definieren wie jedes einzelner dieser Elemente gerendered werden soll in der Liste. Um jeden Artikel darzustellen wurde ein Component `ShoppingItem` geschrieben, welches das `item` der Liste entgegennimmt.

```
<FlatList
    data={this.state.shoppingList}
    renderItem={({item}) =>
        <ShoppingItem item={item}/>
    }
>
```

ShoppingItem ist ein `touchable`, wenn einmal gedrückt wird, wird indiziert dass der Artikel gekauft wurde, bei langem Drücken wird der Artikel aus der Liste gelöscht.

2.4.5 Daten auslesen

Um Daten auszulesen wird eine Callbacke Methode `onCollectionUpdate()` definiert, welche aufgerufen wird sobald `onSnapshot()` gefeuert wird. `onCollectionUpdate()` nimmt einen Snapshot als Parameter, iteriert diesen um die alte Liste mit der neuen zu ersetzen. Wie bereits erwähnt,

wird nur beim ersten Aufruf der gesamte Snapshot der Collection iteriert, nach jedem weiteren Aufruf werden nur mehr die Änderungen verarbeitet.

Um die interne Speicherung zu erleichtern, wird zusätzlich zum Inhalt jedes Dokuments in einer Liste noch das gesamte Dokument und eine eindeutig identifizierbarer Schlüssel gespeichert:

```
    snap.forEach((el) => {
        let { title, bought} = el.data()
        let key : number = el.id
        shoppingList.push({
            key: key,
            doc: el,
            title,
            bought
        });
    });
```

2.4.6 Daten löschen

Wie bereits erwähnt, wird beim langen Drücken eines ShoppingItems der Artikel gelöscht. Dafür wurde im ShoppingItem die Funktion `delete()` geschrieben, welche das gesamte Dokument, welches als Prop an die Komponente übergeben wurde, löscht:

```
delete() {
    this.props.item.doc.ref.delete()
}
```

2.4.7 Daten ändern

Um zu indizieren dass ein Artikel gekauft wurde, wurde das Attribut `bought` im Dokument definiert. Beim einmaligen Drücken eines Artikels, wird die Funktion `toggleBought()` aufgerufen. Diese setzt das Attribut `bought` in der Datenbank auf das Gegenteil welches es davor war:

```
this.props.item.doc.ref.update({
    bought: !this.props.item.bought,
});
```

Zusätzlich wurde noch eine Animation hinzugefügt. Diese Animation lässt einen Artikel innerhalb von 500 Millisekunden Grün erscheinen, sobald dieser gekauft wurde. Falls er, aus welchem Gründen auch immer, doch nicht gekauft wurde, wechselt die Farbe wieder innerhalb von 500ms zu weiß zurück. Dies wurde mit dem `Animated` Modul von React Native umgesetzt:

```
Animated.timing(this.animatedValue, {
    toValue: this.props.item.bought ? 0 : 150,
    duration: 500
}).start()
```

2.4.8 Replikation zur Konsistenzwahrung

Dadurch dass Cloud Firestore die Infrastruktur der Google Cloud Platform verwendet, werden die Daten über mehrere Regionen repliziert und starke Konsistenz-Garantie ist gegeben.[2, Abs. Key capabilities]

2.4.9 Offline-Verfügbarkeit

Offline Verfügbarkeit ist automatisch in Cloud Firestore aktiviert. Hierbei wird eine gecachede Kopie der Daten auf dem Gerät gespeichert. Diese Daten können offline geändert werden und wenn das Gerät wieder online ist, werden diese Daten synchronisiert. [11]

2.4.10 Öffentlich ansprechbar

Dadurch dass es ein Google Service ist, ist Cloud Firestore öffentlich ansprechbar.

2.4.11 Kosten

Der Service ist gratis ist jedoch folgendermaßen limitiert:

Cloud Firestore	
Stored data	1 GB total
Bandwidth	10GB/month
Document writes	20K/day
Document reads	50K/day
Document deletes	20K/day

Abbildung 4: Limitierungen bei der Gratis-Version von Firebase

Literatur

[1] Firebase. https://firebase.google.com/products/. (Accessed on 2018-04-17).

[2] Cloud firestore | firebase. https://firebase.google.com/docs/firestore/. (Accessed on 2018-04-17).

[3] Choose a database: Cloud firestore or realtime database | firebase. https://firebase.google.com/docs/database/rtdb-vs-firestore. (Accessed on 2018-04-17).

[4] Cloud firestore data model | firebase. https://firebase.google.com/docs/firestore/data-model. (Accessed on 2018-04-17).

[5] Interface: Firestore | firebase. https://firebase.google.com/docs/reference/js/firebase.firestore.Firestore. (Accessed on 2018-04-17).

[6] Couchbase mobile | products. https://www.couchbase.com/products/mobile. (Accessed on 2018-04-17).

[7] deployd. http://docs.deployd.com/docs/getting-started/what-is-deployd.html. (Accessed on 2018-04-17).

[8] Firehose | build realtime web applications. http://firehose.io/. (Accessed on 2018-04-17).

[9] invertase/react-native-firebase-starter: A basic react native app with react-native-firebase pre-integrated so you can get started quickly. https://github.com/invertase/react-native-firebase-starter. (Accessed on 2018-04-17).

[10] Firebase console. https://console.firebase.google.com/u/0/. (Accessed on 2018-04-17).

[11] Enable offline data | firebase. https://firebase.google.com/docs/firestore/manage-data/enable-offline. (Accessed on 2018-04-17).

Abbildungsverzeichnis