JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

Synchronisation bei mobilen Diensten

SYT
5BHIT 2017/18

Florian Jäger

Version 1.0
Note:
Begonnen am 15.April 2018
Betreuer:
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 Ergebnis

3.1 Auswahl der Schnittstelle

Die Schnittstelle wurde nach folgenden Kriterien ausgewählt:

- **Synchronisation der Datenbank**

  Die Datenbank soll möglichst leicht synchronisiert werden können.

- **Kompatibilität mit React Native**

  Eine Kompatibilität ist notwendig, da die Applikation, welche die Schnittstelle verwendet in React Native programmiert wurde.

3.1.1 Cloud Firestore

Cloud Firestore ist ein von Google entwickelter NoSQL-Datenbankdienst, welcher erlaubt Daten zu speichern, synchronisieren und abfragen. Der Dienst ist Teil der Plattform Firebase und befindet sich in der Beta-Phase. [1]

Ein großer Vorteil von Firestore ist, dass die Datenbank automatisch synchronisiert wird und offline verwendbar ist. Ebenfalls existieren sehr viel Starter-Repositories, in welchen die benötigten Module installiert wurden. [1]

3.1.2 Couchbase

Couchbase ist ebenfalls ein dokument-basierender NoSQL-Datenbankdienst, welches ähnliche Möglichkeiten wie Cloud Firestore durch Sync Gateway bietet. [2]

Sync Gateway ist einen Web Gateway Applikation, die APIs, Synchronisation und Data Replikation bietet. [3]

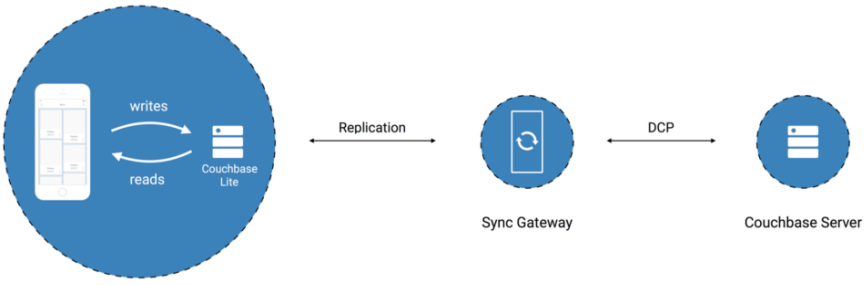

Abbildung 1: Couchbase Sync Gateway [3]

3.1.3 Fazit

Nach Inbetrachtziehung der Kriterien wurde sich für Cloud Firestore entschieden, da die Konfiguration mit Couchbase länger dauern würde und für Firestore ein React Native Repository gefunden wurde.

3.2 Verwendung der Schnittstelle

Um die API von Cloud Firestore zu verwenden wurde das Git-Repository **https://github.com/invertase/re native-firebase-starter[4]** geklont. Dadurch mussten nicht alle Module konfiguriert werden. Nun mussten die Packages heruntergeladen werden mittels dem Befehl **npm install**. Nun kann das Projekt umbenannt werden mit **npm run rename**. Hierbei ist es wichtig den Packagenamen zu merken, da dieser zur Erstellung des Google Service JSON-Files benötigt wird. Dieses kann in der Cloud Firebase Konsole in der Option **Manually add firebase** erstellt und in **/android/app/** gespeicher werden. Nun kann eine Collection in die Applikation eingebunden werden. Dazu wird speziell in den folgenden applikationsspezifischen Punkten eingegangen.

3.3 Applikation

Als Basis für diese Applikation wurde das Tutorial **"Getting started with Cloud Firestore on React Native"[5]** verwendet.

3.3.1 App.js

App.js fungiert als Main-Klasse und rendert das *ShoppingList*-Component.

```
1 export default class App extends Component<{}> {
      render() {
3         return <ShoppingList/>
      }
5 }
```

Listing 1: App-Component

3.3.2 shoppinglist.js

shoppinglist.js beinhaltet das Component *Shoppinglist*. Die Aufgabe des Components ist es die Einkaufsliste inklusive einer Eingabe zu rendern und die Datenbankverbindung zu erstellen bzw. verwalten. Bei dem Start des Konstruktors wird mit der Anweisung eine Referenz zu der Sammlung *shoppinglist* erstellt.

```
1 this.ref = firebase.firestore().collection('shoppinglist');}
```

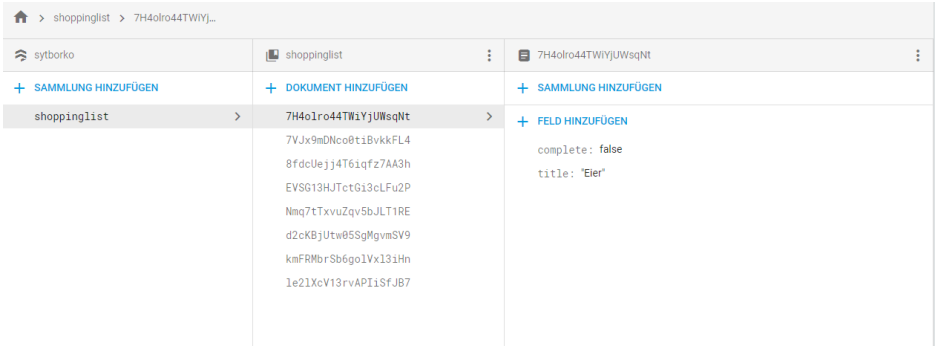

Abbildung 2: Collection in Cloud Firestore

Ebenfalls wird neben einem noch leeren *unsubscribe*-Attribut das Objekt *state* gesetzt, welches durch die verschiedenen Events verändert wird und die Applikation neu-rendert.

```
1 this.unsubscribe = null;
this.state = {
3   textInput: '',
    loading: true,
5   itemList: [],
};
```

Das *textInput*-Attribut wird verändert, wenn es zu einer Texteingabe kommt oder der Add Item Button gedrückt wird. *loading* ist immer **true** wenn die Applikation gerendert wird. In *itemList* befinden sich *ListItem*-Objekte, die in der Liste gerendert werden.

In der *render*-Methode wird die Liste, das Text-Feld und der Button gerendert.

```
  render() {
2   if (this.state.loading) {
      return null;
4   }
    return (
6     <View style={{ flex: 1 }}>
        <FlatList
8       data={this.state.itemList}
        renderItem={({ item }) => <ListItem {...item} />}
10      ItemSeparatorComponent={this.renderSeparator}
        />
12      <TextInput
        placeholder={'Add Item'}
14      value={this.state.textInput}
        onChangeText={(text) => this.updateTextInput(text)}
16      />
        <Button
18      title={'Add Item'}
        disabled={!this.state.textInput.length}
20      onPress={() => this.addItem()}
        />
22    </View>
```

```
        );
24  }
```

Damit nun Waren in die Liste hinzugefügt werden können, wird bei dem Druck des *AddItem*-Buttons die Funktion *addItem()* aufgerufen.

```
    addItem() {
2       this.ref.add({
            title: this.state.textInput,
4           complete: false,
        });
6       this.setState({
            textInput: '',
8       });
    }
```

3.3.3 ListItem.js

In *ListItem.js* wird definiert wie ein Eintrag in der Liste gerendert werden soll.

```
1   render() {
        return (
3           <TouchableHighlight
            onPress={() => this.toggleComplete()}
5           >

7               <View style={{backgroundColor: '#abd0f9', flex: 1, height: 48, flexDirection: 'row',
                    alignItems: 'center' }}>

9                   <View style={{ flex: 8 }}>
                        <Text>{this.props.title}</Text>
11                  </View>

13                  <View style={{ flex: 2 }}>
                    {this.props.complete && (
15                      <Text>Check</Text>
                        )}
17                  </View>
                </View>
19

            </TouchableHighlight>
21      );
    }
```

3.3.4 GUI

Das Endprodukt wird hier dargestellt:

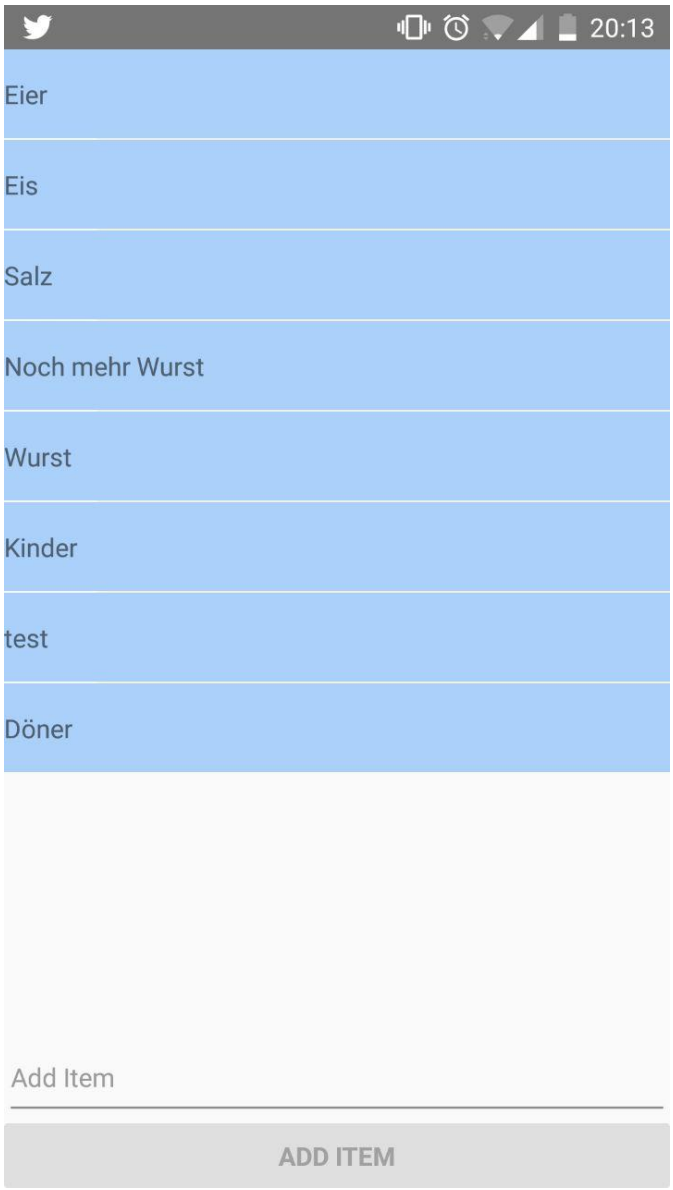

Abbildung 3: App-GUI

4 Anhang

4.1 Abbildungsverzeichnis

4.2 Listingsverzeichnis

4.3 Quellenverzeichnis

[1] Github - 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 04/17/2018).

[2] Nosql engagement database | couchbase. `https://www.couchbase.com/`. (Accessed on 04/17/2018).

[3] Getting comfortable with couchbase mobile: Sync gateway via the command line | the couchbase blog. `https://blog.couchbase.com/getting-comfortable-with-couchbase-mobile-sync-gateway-via-the-command-line/`. (Accessed on 04/17/2018).

[4] 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 04/17/2018).

[5] Getting started with cloud firestore on react native. `https://blog.invertase.io/getting-started-with-cloud-firestore-on-react-native-b338fb6525b9`. (Accessed on 04/17/2018).