JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

SYT

sick

Systemtechnik
5BHIT 2017/18

Johannes Bishara

Version 1.0
Note:
Betreuer: Michael Borko
Begonnen am 18. April 2018
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 Architekturmöglichkeiten

3.1 Client-Server Architektur

Bei der Client-Server Architektur, verbinden sich verschiedene Clients auf einen oder mehreren Servern, um Daten auszutauschen. Man hat dadurch zentrale Instanzen, welche für den Aufbau und Datenaustausch im Netzwerk essenziell sind. Eine der wichtigsten Fragen bei der Client-Server Architektur ist, wie 'wieviel' am Client hinsichtlich der Tiers (View, Application,Data) liegt. Je weniger am Client liegt, desto schneller können auf Änderungen reagiert werden, da die Änderungen nur am Server vorgenommen werden müssen.

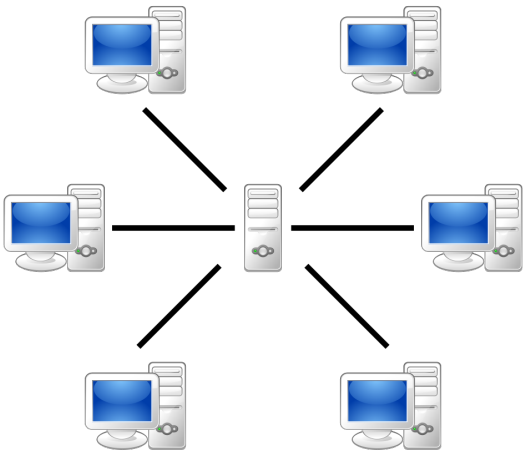

Abbildung 1: Client-Server Architektur

3.1.1 Vorteile

- Performance (Rechenzeit, Speicherverbrauch) auf Server auslagern
- Plattformunabhängig (Business-Topic)

3.1.2 Nachteile

- Single Point of Failure
- Erreichbarkeit (IP,Netzwerk-ID,Online)
- Welche Tiers am Client?

3.2 Peer-to-Peer Architektur

Bei einer Peer-to-Peer Architektur gibt es keine zentrale Instanz. Die Daten werden zwischen den einzelnen Clients ausgetauscht und jeder hält die Daten lokal auf seinem eigenen Rechner.

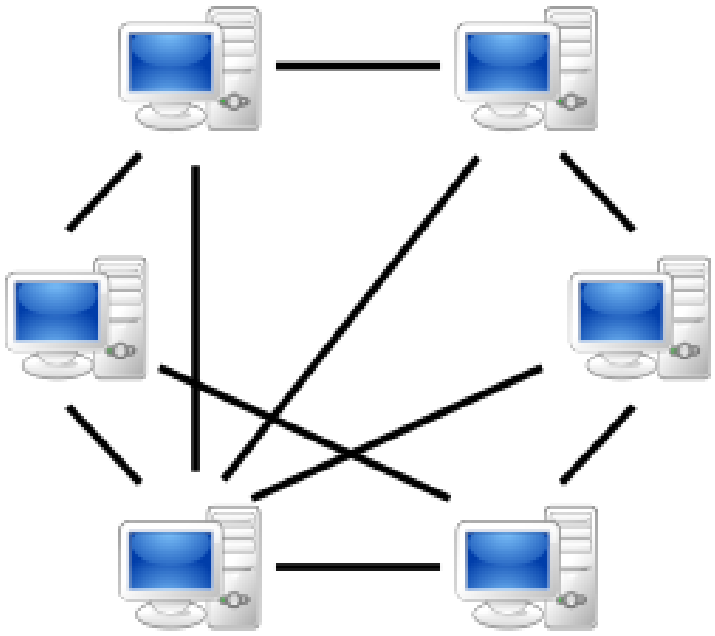

Abbildung 2: Peer-to-Peer Architektur

3.2.1 Vorteile

- Skalierbarkeit erfolgt automatisch
- Verfügbarkeit

3.2.2 Nachteile

- Interkommunikation (zentrale Authentifizierung fehlt)
- Deployment muss zentralisiert sein

3.3 Gewählte Architektur

Es wurde auf eine Client-Server Architektur gesetzt, da bei einer Client-Server ein zentraler Server zur Verfügung steht. Da sich alle Clients eine Einkaufsliste teilen, würde eine zentrale Datenbank Sinn ergeben.

Bei einer Peer-to-Peer Architektur würden Daten die nur lokal auf einem Client liegen. Hat dieser eine Änderung vorgenommen, wäre der Traffic, da dieser mit jedem Client kommunizieren muss anstatt mit einer zentralen Instanz, sehr hoch.

4 Gewählte Datenbank - Firebase

Firebase ist eine Real-Time Datenbank, die kostenlos von Google zur Verfügung gestellt wird. Firebase synchronisiert die Daten automatisch und unterstützt eine offline Synchronisation. Die Daten werden als JSON-Files abgespeichert. Firebase unterstützt Android und IOS Applikationen, sowie Webapplikationen.

Es werden APIs für alle gängigen Programmiersprachen angeboten.

5 Alternativen

5.1 Couchbase Mobile

Genau wie Firebase, bietet Couchbase Mobile eine Datenbank zur Verfügung, welche automatisch Daten zwischen den Clients synchronisiert. Couchbase speichert die Daten in JSON-Files ab.

Hier werden ebenfalls APIs für die gängigen Programmiersprachen angeboten.

5.2 RethinkDB

Es werden Daten über JSON gespeichert. Es eignet sich für das Builden von skalierbaren mobilen Applikationen. Die Verbindung über das Web funktioniert sehr gut, da RethinkDB direkt über das HTTP's Request-Response gemapped wird.

RealthinkDB unterscheidet sich zu Firebase in 3 grundlegenden Punkten:

- Die API ist Open-Source. Es kann ohne Restriktionen in der eigenen Infrastruktur deployed werden.
- Realtime Sync APIs sind auf das synchronisieren von Dokumenten beschränkt, während RethinkDB eine "general purpose Databaseïst.
- RethinkDB wurde designed um direkt von einer Applikation-Server aufgerufen zu werden, anstatt vom Web

6 Umsetzung

Es wurde generell folgendes Tutorial umgesetzt **https://inducesmile.com/android/a-simple-android-todo-list-app-with-recyclerview-and-firebase-real-time-database/**. Es wird nicht

auf das Tutorial generell eingegangen, sondern die Basics erklärt wie man Firebase grundlegend verwenden kann.

6.1 Firebase einrichten

Auf der Seite von Firebase ist es möglich ein neues Projekt einzurichten. Man kann nun ein google-services.json runterladen und dieses in das Android Projekt einfügen. Dieses JSON beinhaltet die Konfiguration der Firebase-Datenbank.

Im Gradle Build-File muss man google-services und Firebase als Dependency hinzufügen und dieses service anschließend applyen.

In den Einstellungen von Firebase wurde der Einfachkeit halber, die Datenbankrechte für das Lesen und Schreiben auf true gesetzt. Somit könnte sich jeder auf die Datenbank verbinden, der die Konfiguration der DB kennt.

In Java muss man um Daten zu pushen lediglich eine Instanz der DB erstellen und pushen:

```
1 databaseReference = FirebaseDatabase.getInstance().getReference();

3 databaseReference.push().setValue(taskObject);
```

Datensätze können folgendermaßen gelöscht werden:

```
1 String taskTitle = singleSnapshot.getValue(String.class);
  for(int i = 0; i < allTask.size(); i++){
3     if(allTask.get(i).getTask().equals(taskTitle)){
          allTask.remove(i);
5     }
  }
```

Literatur

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

