JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Distributed Computing
GK10.2 Mobile Dienste (BORM)

Systemtechnik Labor
5BHIT 2017/18

Pierre Rieger

Version 1.0
Begonnen am 15. März 2018
Beendet am 18. April 2018

Note:
Betreuer: Michael Borko

Inhaltsverzeichnis

1 Einführung **1**

2 Ergebnisse **2**

2.1 Der Server 2

2.1.1 Implementierung 2

2.2 Die App 3

2.2.1 Implementierung 3

2.3 Beschreibung des Synchronisationsansatzes und Design der gewählten Architektur (Interaktion, Datenhaltung) 3

2.4 Recherche möglicher Systeme bzw. Frameworks zur Synchronisation und Replikation der Daten 4

2.5 Dokumentation der gewählten Schnittstellen 4

2.6 Implementierung der gewählten Umgebung auf lokalem System 4

2.7 CRUD Implementierung u. Implementierung eines Replikationsansatzes zur Konsistenzwahrung 4

2.8 Offline-Verfügbarkeit 4

2.9 System global erreichbar 4

1 Einführung

Es soll eine App erstellt werden, die in der Lage ist eine einfache Shoppingliste zu verwalten. Dabei sollen mehre Clients zu einer Liste verbunden werden, dies sollen in der Lage sein Elemente einzufügen und zu Löschen.

2 Ergebnisse

2.1 Der Server

Der Server muss in der Lage sein mehrere Nutzer zu verwalten und diese einer Liste zuzuweisen. Um mehrere verschieden Listen zu verwalten wird jede Liste als Topic unterteilt. Jedes Topic besitzt einen eigenen Zustand der den derzeitigen Inhalt der Liste enthält. Darüber hinaus hat jedes Topic einen Namen und ein Passwort. Sobald der Nutzer einem Topic beitritt erhält er den Zustand der derzeitigen Daten. Der Client kann dann Daten senden an den Server senden. Das könnte allerdings mit einem Algorithmus wie Diffsync noch performanter werden. Zurzeit verwendet der Server allerdings ein einfaches String Array.

2.1.1 Implementierung

Der Server ist als Node.js Server konzipiert, er verwendet Express und Websockets für die Kommunikation. Bei Websockets ist es wichtig viel Aufwand in Fehlerbehandlung zu legen. Ein Hearbeat Ping/Pong stellt sicher, dass keine 'toten Sockets existieren', Flow gibt Typsicherheit um gängige Programmierfehler zu vermeiden. In dieser Version können allerdings noch einige Fehler auftreten falls unerwartet Requests gemacht werden.

Der Server gliedert jede Einkaufsliste als einzelnes Topic, jedes Topic enthält eine Liste an Verbindungen und eine Liste für alle Daten die das Topic verwaltet. Ein Topic informiert alle Verbindungen sobald eine Veränderung auftritt und sendet die gesamte Datenliste. Ein weitere Verbesserung wäre es, wenn nur die Änderungen gesendet werden würden.

Der Server Antwortet im Datenformat JSON eine erfolgreiche notification schaut dann in etwa folgendermaßen aus:

```
{
 type:'data',
 status: 'data',
 success: true,
 data: [ 'eier', 'milch', 'mehl' ],
 topic: 'topic'
}
```


2.2 Die App

Die App ist in React-Native entworfen und baut mittels Websockets eine Verbindung zum Server auf. Ein Client kann Topics auf dem Server erzeugen und ihnen beitreten, im Moment hat der Client kein 'Gedächtnis', er merkt sich also nicht in welchen Listen er bereits ist, darauf könnte in Zukunft aufgebaut werden indem der Client zum Beispiel RNFS (React-Native-File-System) verwendet um die Topic Namen zu speichern. Wenn der Client ein Topic erzeugen will dann sendet er eine JSON Request an den Server mit dem Wert `action: "new"` daraufhin bekommt er eine Rückmeldung ob die Erstellung erfolgreich war oder nicht. Im Moment werden misslungenge Erstellungen nicht gehandhabt. Das Beitreten läuft ähnlich ab, genauso wie das Notifizieren.

2.2.1 Implementierung

Die Network Funktionalitäten werden getrennt vom GUI Code gehandhabt, das GUI binded einzelne Listener an den Networker und initialisiert ihn sobald alle Funktionen gebunden sind. Der Networker ist ebenfalls als Singleton entworfen und bietet einige Funktionen um Daten, im vom Server erwarteten Format, zu senden. Beispielsweise werden die Daten am Server durch folgenden Code aktualisiert:

```
sendUpdateData(data : string[]) {
    const name = this.name
    const pw = this.pw
    if (!this.ws) return
    this.ws.send(JSON.stringify({
      action: "notify",
      topic: name,
      topicPass : pw,
      data: data
    }))
}
```

Listing 1: aktualiseren der Daten

2.3 Beschreibung des Synchronisationsansatzes und Design der gewählten Architektur (Interaktion, Datenhaltung)

Die Synchronisation erfolgt mittels Websockets, sobald eine Sitzung aufgebaut wird, erhält der Client den gesamten Zustand der Liste, bei jeder Veränderung wird ebenfalls die gesamte Liste gesendet. Eine Möglichkeit um die Performanz zu erhöhen wäre ein Algorithmus der lediglich die veränderten Daten sendet. Jeder Client ist in Topcis eingegliedert, jedes Topic beinhaltet die Daten. Wenn ein Client eine Veränderung an einer Liste vornimmt, werden alle anderen Clients des selben Topics informiert.

2.4 Recherche möglicher Systeme bzw. Frameworks zur Synchronisation und Replikation der Daten

Wie oben schon erwähnt alternativ könnte DiffSync verwendet werden bzw. Firebase, diese Frameworks sind relativ robust.

2.5 Dokumentation der gewählten Schnittstellen

Siehe 2.1 und 2.2

2.6 Implementierung der gewählten Umgebung auf lokalem System

Mittels React-Native kann die App auf einem Emulator gestartet werden, es ist ebenfalls möglich die App direkt auf ein Handy zu spielen bzw. die App für Android zu builden und dann mittels `adb install` auf einem mobilen Gerät zu installieren.

2.7 CRUD Implementierung u. Implementierung eines Replikationsansatzes zur Konsistenzwahrung

siehe 2.1.1

2.8 Offline-Verfügbarkeit

Ist derzeit nicht implementiert, mit Tools wie Firebase wäre es leicht möglich diese Funktion hinzuzufügen. Andernfalls kann RNFS verwendet werden um ein lokales File zu erstellen, mit dem man den derzeitigen bzw. den letzten Zustand speichern könnte.

2.9 System global erreichbar

Das System ist auf Heroku bereitgestellt und ist damit öffentlich erreichbar.

Listings

1 aktualiseren der Daten 3