SYT

5BHIT 2017/18

Synchronisation bei mobilen Diensten

Ausarbeitung

Colakovic Zeljko

18. April 2018

Bewertung:		Version:	1
Betreuer:	Borko Michael	Begonnen:	10.02.2018
		Beendet:	18.04.2018

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 Vorraussetzung

- 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 d es 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 Ërweiterte-Kompetenz überwiegend erfüllt"**
- CRUD Implementierung
- Implementierung eines Replikationsansatzes zur Konsistenzwahrung

- **Anforderungen Ërweiterte-Kompetenz zur Gänze erfüllt**"
- Offline-Verfügbarkeit
- System global erreichbar

2 Abgabe

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

3 Vergleich der Systeme

Für die Auswahl des Systems wurde zwischen Diffsync, Couchbase Mobile und Firebase abgewogen.

Dabei bietet Firebase einige Vorteile wie das keine Realtime keine Sockets notwendig sind. Ebenfalls Keine REST API selber aufgesetzt werden. Dazu ist es eine kostengünstige Variante. Die Nachteile hingegen sind beispielsweise, dass Firebase nicht selber gehostet werden kann und nicht für Big Data geignet ist. Außerdem ist es lokal nicht ausführbar sondern nur in der Cloud.

Couchbase Mobile bietet Vorteile wie niedrige Kosten und das es nicht nur in der Cloud sondern auch lokal gehostet werden kann. Jedoch hat Couchbase Mobile probleme mit der Synchronisation von Daten welche vom Client auf den Server geschoben werden müssen.

Bei Diffsync sind die Vorteile, die einfache einrichtung des Systems sowie, dass keine Datenbank von notwendigkeit ist. Die Nachteile hingegen ist, dass es keine Offlineverfügbarkeit bietet. Sowie die notwendigkeit eines Servers.

Aufgrund von einem vorhandenen Tamplate welches nur erweitert werden musste, wurde Diffsync gewählt.

4 Durchführung

4.1 Diffsync

4.1.1 Funktionsweise und Umsetzung

Durch Diffsync werden Daten (JSON-Objekte) mittels Websockets synchronisiert. Durch den Befehl npm install diffsync kann Diffsync einfach installiert werden.

Für die Aufgabe wurde ein Client sowie ein Server erstellt, dabei wurde das SYT-Beispiel von Kevin Waldock verwendet und erweitert. Der Server benötigt hier eine socket.io-Instanz, außerdem einen dataAdapter-Objekt um die Daten zu verarbeiten (empfangen, speichern, synchronisieren).

```
// setting up socket.io
let io = require('socket.io')(3000, {log: false, origins: '*:*'});

// setting up diffsync's DataAdapter
let diffsync = require('diffsync');
let dataAdapter = new diffsync.InMemoryDataAdapter();

// setting up the diffsync server
let diffSyncServer = new diffsync.Server(dataAdapter, io);
```

Mit Hilfe von von SiffSyncClient-Objekten wird eine Verbidnung zwischen Server und Client erstellt. Für die Synchronisation dient die Funktion client.on('connected',callback()), welche aufgerufen wird wenn sich der Client zum server verbindet. Die Funktion client.on('synced',callback()) wird aufgerufen, wenn synchronisiert wird. Durch die Funktion client.sync();, welche der Client aufruft, wird schlussendlich synchronisiert.

```
let data;
client.on('connected', function(){
    // Die Datenreferenz, welche zum synchronisieren verwendet wird.
    // Data-Obj wird sync
    data = client.getData();
    console.log('Verbunden!');
    console.log('Daten akutell:');
    console.log(data);
    document.getElementById("button").addEventListener('click', () =>
    ↪  addElements());
    data['form'] = [];
});
```

Um das Beispiel von Kevin auf Windows lauffähig zu gestalten müssen einige File geändert werden. Zuerst im server/index.js der socket.io ersetzt werden:

```
// setting up socket.io
let io = require('socket.io')(3000, {log: false, origins: '*:*'});
```

Im client/package.json muss ebenfalls eine Änderung vorgenommen werden. Hier wird ßcripts": mit folgendem Code ersetzt. Damit wird ein Http-Service mit dem Client gestartet.

```
  "scripts": {
    "start": "webpack src/index.js dist/bundle.js",
    "server": "http-server dist"
  },
```

Im File client/dist/index.js wurden eigene Funktionen implementiert. Durch das Erzeugen eines Arrays (subitem des data-Objekt) im client.on, werden die Daten in diesem abgespeichert.

```
//Durch Buttonklick wird abgespeichert
document.getElementById("button").addEventListener('click', () =>
↪  addElements());

//Subitem zum abspeichern der Daten
data['form'] = [];
```

Durch die Funktion addElements werden über die HTML-Seite die Werte im data-Array gespeichert.

```
function addElements(){

    // Abspeichern der vom User eingegebenen Werte ins Array
    data['form'].push({"Produkt" :
    ↪  document.getElementById("form-produkt").value, "Menge" :
    ↪  document.getElementById("form-menge").value, "Note" :
    ↪  document.getElementById("form-note").value};

    client.sync();
}
```

Für die Ausgabe im HTML dient die Funktion output.

```
function output(){
    var str = "";

    // Nimmt die Elemente aus dem data-Array und schreibt es in einem
    ↪  String
    for (var i in data['form']){
        str += "<b>Produkt:</b>" + data['form'][i]["Produkt"]+ "
        ↪  <b>Menge:</b>" + data['form'][i]["Menge"] + " <b>Note:</b>"
        ↪  + data['form'][i]["Note"] + "<br>";
    }

    // Sorgt für die Ausgabe im HTML
    document.getElementById("output").innerHTML = str;
}
```

4.1.2 Starten der Applikation

Zum starten der Applikation öffnet man zunächst eine CMD und begibt sich in den Server-Folder, wo folgenden Befehl ausführt:

```
npm start
```

In einer zweiten CMD begibt man sich in den Client-Folder, wo man folgenden Befehl ausführt:

```
npm run server
```

Darauf hin ruft man localhost:8080 im Browser auf.

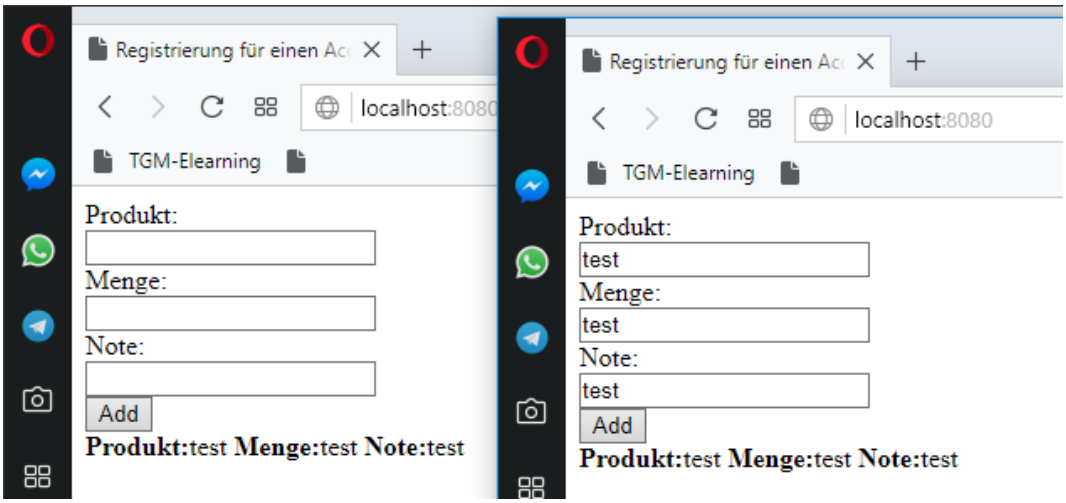

Abbildung 1: Applikation

5 Quellen

- https://firebase.google.com/docs/ [Zuletzt aufgerufen: 18.04.2018]
- http://mfg.fhstp.ac.at/development/webdevelopment/firebase-eine-kurze-einfuehrung/ [Zuletzt aufgerufen: 18.04.2018]
- https://stackoverflow.com/ [Zuletzt aufgerufen: 18.04.2018]
- https://www.golem.de/1201/88815.html [Zuletzt aufgerufen: 18.04.2018]
- https://www.couchbase.com/products/mobile [Zuletzt aufgerufen: 18.04.2018]
- https://github.com/KevinW1998/diffsync-template [Zuletzt aufgerufen: 18.04.2018]

Abbildungsverzeichnis