JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Arbeitsprotokoll

Synchronisation bei mobilen Diensten

Systemtechnik
5BHIT 2017/18

Hamberger Gregor

Version 1

Note:
Betreuer:

Begonnen am 17. 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 Ergebnisse

2.1 Apache Cordova

Apache Cordova ist ein Open-Source Framework zur plattformübergreifenden Programmierung von mobilen Applikationen mit Hilfe von JavaScript, HTML5 und CSS3.[1]

Die Vorteile liegen dabei bei der Möglichkeit mit Hilfe einer einzigen Codebase für mehrere Plattformen entwickeln zu können. Durch die Nutzung von *npm* als Paketmanager ist auch die Suche und das Einbinden von weiteren notwendigen Frameworks wesentlich vereinfacht. Es ist damit möglich auf bereits bekannte Lösungen zurückzugreifen, was den Prozess der Entwicklung noch einmal beschleunigt.

Die Nachteile liegen wie bei allen plattformübergreifenden System in einer etwas schlechteren Optimierung der Darstellung. Die Systemressourcen können unter Umständen nicht optimal ausgenutzt werden, was zum Beispiel zu Ruckeln in der grafischen Oberfläche führen kann.

2.2 node-couchdb

Das *node-couchdb* Paket[2] ist ein *npm* Paket zur Interaktion mit CouchDB selbst[3]. Es vereinfacht den Netzwerkzugriff und die Bedienung der genutzten CouchDB Datenbank. CouchDB selbst ist eine dokumentenorientierte Datenbank und bietet eine HTTP/JSON Schnittstelle, wodurch sie sich für unsere Zwecke gut eignet.
Die Installation erfolgt mit Hilfe von *npm* denkbar einfach:

```
npm install node-couchdb -save
```

Nun kann das Paket direkt genutzt werden. Der Konstruktor nimmt ein Objekt entgegen, in welchem alle notwendigen Parameter für eine Verbindung zur Datenbank vorhanden sein können (aber nicht müssen):

```
1 //Beispiel zur Verbindung mit einer remote Datenbank
const couchExternal = new NodeCouchDb({
3     host: 'couchdb.external.service',
      protocol: 'https',
5     port: 6984
});
```

Folgende Optionen stehen hierbei zur Verfügung:

- **host** Netzwerkadresse des Datenbankservers (Default: 127.0.0.1)
- **port** Port auf den CouchDB hört (Default: 5984)
- **protocol** Zu verwendendes Protokoll (Default: HTTP)
- **cache** Sollte ein cache Plugin wie z.B. *memcached* verwendet werden ließe sich dieses hier spezifizieren (Default: null)

- **auth** Objekt mit Benutzername und Password
- **timeout** Zeit in Millisekunden bevor ein Timeout angezeigt wird (Default: 5000)

Dann lassen sich die Dokumente in der Datenbank manipulieren.

```
//Erstellen eines neuen Dokuments
couch.insert("databaseName", {
    _id: "document_id",
    field: ["sample", "data", true]
}).then(({data, headers, status}) => {
    // data is json response
    // headers is an object with all response headers
    // status is statusCode number
}, err => {
    // either request error occured
    // ...or err.code=EDOCCONFLICT if document with the same id already exists
});

//Einfuegen von Daten
couch.update("databaseName", {
    _id: "document_id",
    _rev: "1-xxx"
    field: "new sample data",
    field2: 1
}).then(({data, headers, status}) => {
    // data is json response
    // headers is an object with all response headers
    // status is statusCode number
}, err => {
    // either request error occured
    // ...or err.code=EFIELDMISSING if either _id or _rev fields are missing
});
```

2.3 Diffsync

Diffsync[4] ermöglicht JSON Objekte kollaborativ zu editieren. Die Installation erfolgt auch hier wieder mit *npm*:

```
npm install pouchdb
```

2.3.1 Server

Um nun Daten senden und empfangen zu können wird ein `dataAdapter` Objekt benötigt. Dafür wiederum muss zuerst eine `socket.io` Instanz erzeugt werden:

```
// socket.io Instanz erzeugen
let io = require('socket.io');

// DataAdapter Objekt erstellen
let diffsync = require('diffsync');
let dataAdapter = new diffsync.InMemoryDataAdapter();

// Anschliessend kann der Diffsync Server erzeugt werden
let diffSyncServer = new diffsync.Server(dataAdapter, io);
```

2.3.2 Client

Um sich nun auf den eben erstellten Server verbinden zu können, wird der Client wie folgt implementiert:

```
let DiffSyncClient = require('diffsync').Client;
let socket = require('socket.io-client');
let client = new DiffSyncClient(socket('http://localhost:3000'), 'form-data');
```

Anschließend wird vor allem die `client.on()` Methode verwendet. Diese nimmt als ersten Parameter verschiedene Status entgegen und ermöglicht so auf diese zu reagieren. So wird zum Beispiel der Status `connected` genutzt um unsere Einkaufsliste beim ersten Verbinden zu lesen.

```
client.on('connected', function(){
    data = client.getData();
});
```

Dann kann mit dem Status `synced` jedes Mal eine Operation ausgeführt werden wenn der Server neue Informationen pusht.

```
client.on('synced', function(){
    // GUI updaten, etc.
});
```

2.4 Datenaustausch

Der Datenaustausch erfolgt wie bereits erwähnt in Form von JSON Dokumenten. Der allgemeine Aufbau ist hierbei möglichst einfach gehalten:

```
{
    "liste":{
        "<ListeNr>":{
            "name":"<String>",
            "menge":<Integer>
        }
        "<ListeNr>":{
            "name":"<String>",
            "menge":<Integer>
        }
    }
}
```

Literatur

[1] Apache Foundation. Apache Cordova. `https://cordova.apache.org/`, 2018. [Online; accessed 17-April-2018].

[2] Inc. npm. node-couchdb. `https://www.npmjs.com/package/node-couchdb`, 2018. [Online; accessed 17-April-2018].

[3] Apache Foundation. node-couchdb. `http://couchdb.apache.org/`, 2018. [Online; accessed 17-April-2018].

[4] Inc. npm. node-couchdb. `https://www.npmjs.com/package/node-couchdb`, 2018. [Online; accessed 17-April-2018].

Tabellenverzeichnis

Listings

Abbildungsverzeichnis