JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Systemtechnik Theorie
Synchronisation bei mobilen Diensten

Systemtechnik
5BHIT 2017/18

Jakub Kopanski

Version 1
Note:
Begonnen am 14. April 2018
Betreuer: Prof. Borko
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 Vorgehensweise

Es wurde folgendes Tutorial[1] als Basis für die Aufgabenstellung verwendet:

Create Offline Web Apps Using Service Workers & PouchDB

Die fertige Einkaufsliste als Web-App ist hier zu finden: `www.github.com/jkopanski2/einkaufsliste`

Als Einstieg in dieses Thema folgt zuerst ein Abschnitt mit allgemeinen Erklärungen.

4 Offline-First Applikationen

Sogenannte Offline-First[2] Applikationen funktionieren auch, wenn sie keine oder auch nur eine schlechte Internetverbindung haben. Durch das cachen der Seite ist es möglich auf diese auch offline zuzugreifen. Die Daten werden in einer lokalen Datenbank gespeichert, die sich bei einer bestehenden Internetverbindung wieder mit einer anderen Datenbank im Backend synchronisiert. [3] Somit kann der Benutzer beispielsweise immernoch Einträge in der Einkaufsliste tätigen, auch wenn er einmal keine Verbindung zum Internet hat.

4.1 Mögliche Technologien

4.1.1 Couchbase Sync Gateway

Couchbase hat eine Möglichkeit die mobile Datenbank *Couchbase Lite* mithilfe eines *Couchbase Sync Gateway* mit einem *Couchbase Server* zu synchronisieren. Die Schnittstelle liefert die Daten vom Client an den Server und sorgt somit für eine Sicherung der Daten.

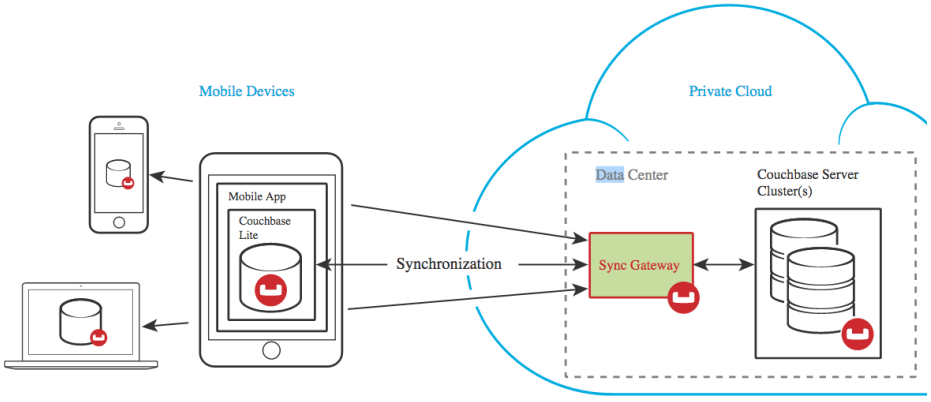

Abbildung 1: Couchbase Sync Gateway[4]

Hier eine Beispielanwendung, welche die Synchronisation mittels dem *Couchbase Sync Gateway* und die Offline Verfügbarkeit mittels *PouchDB*, welches später erklärt wird, zeigt: Using PouchDB and Couchbase in an Offline-First Application[5]

4.2 Firebase Realtime Database

Mithilfe der Google Plattform Firebase kann man ebenfalls Daten in seiner Web Applikation synchronisieren. Hierfür wird die *Realtime Database*[6] verwendet, welche die Daten über alle Endgeräte mit der NoSQL Datenbank synchronisiert. Eine Beispielanwendung, welche genau den Anforderungen dieser Übung entspricht, ist hier zu finden: Firebase CRUD Web App with Javascript[7]

4.3 CouchDB - PouchDB mit Service Worker

Als lokale Datenbank wird hier die dokementenbasierte Datenbank *PouchDB* verwendet, welche sich mit *CouchDB* synchronisieren lässt. Dies ist problemlos möglich, da *PouchDB* auf dem *CouchDB sync Protkoll*[8] basiert.[9] Für das Caching ist ein *Service Worker*[10] zuständig. Dieser ist zwischen Server und Browser und kann somit steuern, welche Inhalte angezeigt werden sollen. Bei einer mangelnden Internetverbindung liest er die gecachten Daten aus und ermöglicht dem Benutzer somit die Offline Verfügbarkeit.

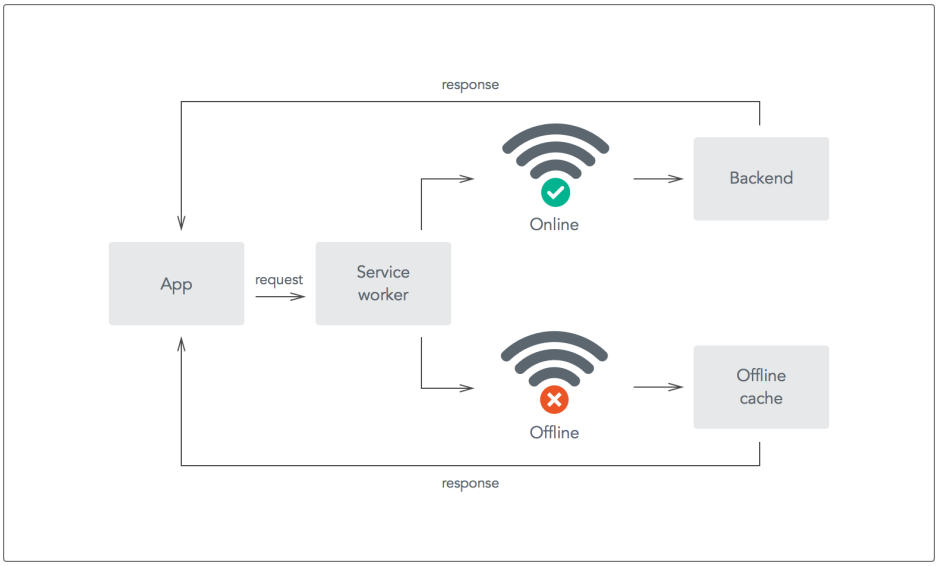

Abbildung 2: Service Worker[11]

4.4 Gewählte Technologie

Durch die fast automatische Replikation der Daten zwischen *PouchDB* und *CouchDB* wurde dieser Ansatz gewählt. In der Hoffnung sich komplexe Konfigurationen, wie es beispielsweise bei *Couchbase* der Fall ist, zu ersparen. Die Google-Lösung mittels Firebase wurde ausgelassen, da man eine „kleinere“ Lösungen ausprobieren wollte.

5 Implementierung

5.1 Service Worker

Zu Beginn wird überprüft ob der *Service Worker* vom Browser unterstützt wird. Wenn ja, wird die Methode zum Registrieren aufgerufen. In diesem Fall ist es die *service-worker.js*, welche aufgerufen wird:

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/service-worker.js', {
        scope: '/'
    }).then(function() {
        // success
    }).catch(function(e) {
        // failed
    });
}
```

Danach wird der *Service Worker* installiert. Hierfür muss unter anderem angegeben werden, welche Files gechacht werden sollen.

```
var CACHE_NAME = 'einkaufsliste';

var resourcesToCache = [
  '/',
  '/css/style.css',
  '/js/ext/babel.min.js',
  '/js/ext/pouchdb.min.js',
  '/js/register-service-worker.js',
  '/js/store.js',
  '/js/app.js'
];

self.addEventListener('install', function(event) {
  event.waitUntil(
    // open the app browser cache
    caches.open(CACHE_NAME)
      .then(function(cache) {
        // add all app assets to the cache
        return cache.addAll(resourcesToCache);
      })
  );
});
```

Zuletzt wird noch festgelegt, wie auf das fetch-event[12] reagiert werden soll, welches bei jedem Request einer Seite ausgelöst wird.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    // try to find corresponding response in the cache
    caches.match(event.request)
      .then(function(response) {
        if (response) {
          // cache hit: return cached result
          return response;
        }

        // not found: fetch resource from the server
        return fetch(event.request);
      })
  );
});
```

Der *Service Worker* wird derzeit noch nicht von allen Browsern unterstützt[13], weswegen zusätzlich noch der AppCache[14] eingerichtet wird.

5.2 PouchDB und CouchDB

Für die lokale Datenspeicherung ist *PouchDB* zuständig. Die Idee dahinter ist, in der Zeit in der keine Internetverbindung besteht, die Daten lokal zu speichern und bei wiederkommender Verbindung diese dann mit *CouchDB* zu synchronisieren.

Hierfür wurden benötigte CRUD-Funktionen implementiert:

```
class Store {

  constructor(name, remote, onChange) {
    this.db = new PouchDB(name);

    // start sync in pull mode
    PouchDB.sync(name, `${remote}/${name}`, {
      live: true,
      retry: true
    }).on('change', info => {
      onChange(info);
    });
  }

  getAll() {
    // get all items from storage including details
    return this.db.allDocs({
        include_docs: true
      })
      .then(db => {
        // re-map rows to collection of items
        return db.rows.map(row => {
          return row.doc;
        });
      });
  }

  get(id) {
    // find item by id
    return this.db.get(id);
  }

  save(item) {
    // add or update an item depending on _id
    return item._id ?
      this.update(item) :
      this.add(item);
  }

  ...
```

Im Hauptprogramm wird nun auf diese Funktionen zugegriffen:

```
class Einkaufsliste {

  constructor(storeClass, remote) {
    this.store = new storeClass('einkaufsliste', remote, () => {
      // refresh contact list when data changed
      this.refresh();
    });

    ...
  }

  showContact(event) {
    // get contact id from the clicked element attributes
    var contactId = event.currentTarget.getAttribute(CONTACT_ID_ATTR_NAME);

    // get contact by id
    this.store.get(contactId).then(contact => {
      // show contact details
```

```
            this.setContactDetails(contact);

            // turn off editing
            this.toggleContactFormEditing(false);
        })
    }

    ...
}
```

Mit der *PouchDB.synch()* Funktion wird eine Synchronisation mit *CouchDB* sichergestellt. Hier wird der Name der lokalen Datenbank und die Zieldatenbank angegeben. In unserem Fall also *PouchDB.sync('einkaufsliste', 'http://localhost:5984/einkaufsliste');*

Hierfür wird eine lokale *CouchDB* Instanz auf dem Port 5984 benötigt, welche zusätzlich noch CORS[15] aktiviert hat. Dies kann man über die grafische Oberfläche unter '*http://localhost:5984/_utils/#_config/couchdb@localhost/cors*' konfigurieren. Für diese Test-Applikation werden alle Domains zugelassen.

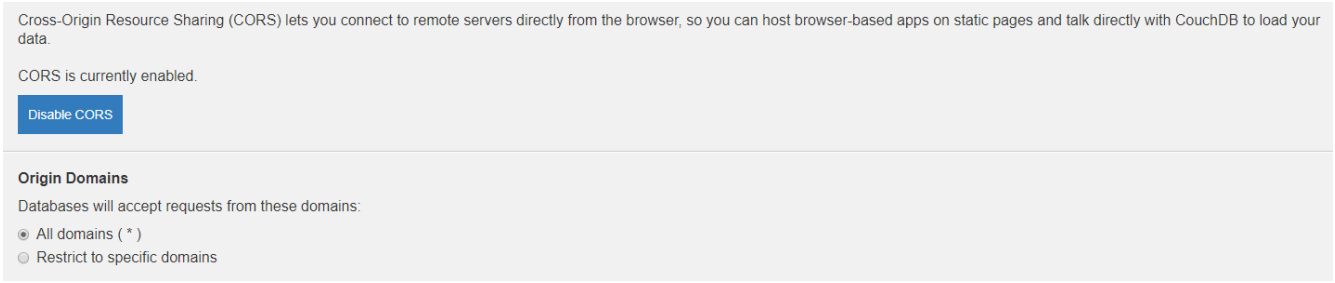

Abbildung 3: CORS Konfiguration CouchDB

Literatur

[1] Artem Tabalin. Create Offline Web Apps Using Service Workers & PouchDB. https://www.sitepoint.com/offline-web-apps-service-workers-pouchdb/, 2017. [Online; accessed 16-04-2018].

[2] Offline First. http://offlinefirst.org/, 2018. [Online; accessed 16-04-2018].

[3] Pedro Teixeira. Build More Reliable Web Apps with Offline-First Principles. https://thenewstack.io/build-better-customer-experience-applications-using-offline-first-princ 2016. [Online; accessed 16-04-2018].

[4] Couchbase. Couchbase Sync Gateway. https://www.couchbase.com/products/sync-gateway, 2018. [Online; accessed 16-04-2018].

[5] Peter Mbanugo. Using PouchDB and Couchbase in an Offline-First Application. https://www.codementor.io/pmbanugo/using-pouchdb-and-couchbase-in-an-offline-first-application-5pw2sxs6o, Mar 08, 2017. [Online; accessed 16-04-2018].

[6] Google. Firebase Realtime Database. https://firebase.google.com/docs/database/, April 13, 2018. [Online; accessed 17-04-2018].

[7] Raja Tamil. Firebase CRUD Web App with Javascript. http://softauthor.com/learn-to-build-firebase-crud-app-with-javascript-part01-reading-data/, Mar 24, 2018. [Online; accessed 17-04-2018].

[8] Apache Software Foundation. CouchDB Replication Protocol. http://docs.couchdb.org/en/master/replication/protocol.html, 2018. [Online; accessed 16-04-2018].

[9] PouchDB. Introduction to PouchDB. https://pouchdb.com/guides/, 2018. [Online; accessed 16-04-2018].

[10] Matt Gaunt. Service Workers: an Introduction . https://developers.google.com/web/fundamentals/primers/service-workers/, Mar 29, 2018. [Online; accessed 16-04-2018].

[11] Ryan Chenkie. Creating Offline-First Web Apps with Service Workers. https://auth0.com/blog/creating-offline-first-web-apps-with-service-workers/, October 30, 2015. [Online; accessed 16-04-2018].

[12] fscholz. FetchEvent. https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent, Feb 13, 2018. [Online; accessed 17-04-2018].

[13] Service Worker Browser Support. https://caniuse.com/#feat=serviceworkers, 17-04-2018. [Online; accessed 17-04-2018].

[14] Eric Bidelman. A Beginner's Guide to Using the Application Cache. https://www.html5rocks.com/en/tutorials/appcache/beginner/, June 18, 2010. [Online; accessed 17-04-2018].

[15] The Apache Software Foundation. CouchDB CORS. http://docs.couchdb.org/en/1.3.0/cors.html, 2013. [Online; accessed 17-04-2018].

Tabellenverzeichnis

Listings

List of Listings

Abbildungsverzeichnis