Studiengangsspezifisches Deckblatt

Das studiengangsspezifische Deckblatt ist in der Materialiensammlung enthalten.
Das Deckblatt kann separat ausgefüllt und ausgedruckt werden.

Kurzfassung

TODO

English Title of the Thesis

Abstract

TODO

Ehrenwörtliche Erklärung

Hiermit erkläre ich, **Philipp Reinking**, geboren am **21. Januar 1989 in Marl**, ehrenwörtlich, dass ich meine Diplom-/Bachelor-/Masterarbeit mit dem Titel:

„*Entwicklung einer Social Extranet Plattform als hybride Applikation für mobile Endgeräte mit Online und Offline Synchronisierung*“

selbstständig und ohne fremde Hilfe angefertigt und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe.

Die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen habe ich innerhalb der Arbeit gekennzeichnet.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Zweibrücken, 14. April 2014

Sperrvermerk

Die vorliegende Diplom-/Bachelor-/Masterarbeit enthält vertrauliche Daten und Informationen der **Pixelpark AG**. Veröffentlichungen oder Vervielfältigungen -auch nur auszugsweise - sind ohne ausdrückliche schriftliche Genehmigung des Unternehmens nicht gestattet.

Die Arbeit ist nur den Korrektoren sowie erforderlichenfalls den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Soziale Netzwerke haben sich in den letzten Jahren zu einem alltäglichen Instrument der Kommunikation und Informationsbeschaffung entwickelt. Menschen können mithilfe solcher Netzwerke sehr einfach miteinander in Kontakt treten, sich vernetzen und organisieren.
Auch das Nachrichtenwesen befindet sich zur Zeit im Wandel, denn Informationen verbreiten sich auf Twitter oder Facebook meist schneller, als über klassische Medien wie TV, Radio oder Print. Die oftmals als "viral" bezeichnete Ausbreitung von Beiträgen ist ein Phänomen das man sehr häufig in sozialen Netzwerken beobachten kann und von dem viele Personen, Organisationen und Unternehmen zu profitieren versuchen, besonders wenn es um die Reichweite von Informationen und Kommunikation geht.

Ein soziales Netzwerk, in welchem sich Menschen miteinander vernetzen können, ob als Extranet einer Organisation oder als öffentliches Netzwerk, wird heutzutage von einer Vielzahl von Faktoren bestimmt und beeinflusst.

Ein vor allem entscheidender Faktor in einem Netzwerk ist die Orientierung und Ausrichtung an einem bestimmten Thema oder Themenbereich. Und immer häufiger entstehen auf solche sehr beschränkte Themenbereiche spezialisierte Plattformen, die unter Umständen zwar weniger Personengruppen ansprechen können, aber in der grundlegenden Funktionsweise das Ziel dieses Netzwerks besser unterstützen. (Beispiele: dribbble.com, stackoverflow.com, github.com).

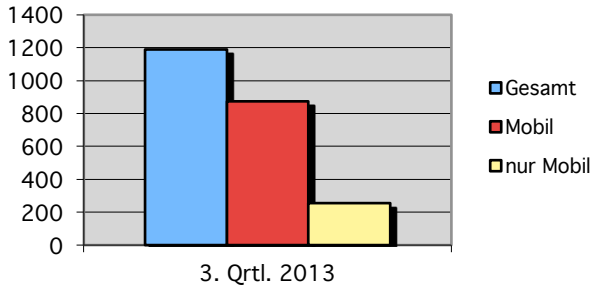

Abbildung 1 - Facebook Nutzerzahlen 2013

Auch die mobile Internetnutzung muss als weiterer wichtiger Faktor genannt werden, dem immer größere Bedeutung zugesprochen wird. Facebook liefert beispielsweise interessante Statistiken dazu (siehe Abbildung 1). So haben ca. 73% der monatlich aktiven Nutzer das Netzwerk auch von einem mobilen Endgerät genutzt. Zusätzlich nutzten ca. 21% der Nutzer das Netzwerk ausschließlich über ein mobiles Gerät. Ein Grund für die ausschließliche mobile Nutzung kann das Fehlen der entsprechenden Infrastruktur für einen festen Internetanschluss sein, insbesondere in Entwicklungsländern.

Zusätzlich sind Sicherheit, Datenschutz und Privatsphäre gerade zur heutigen Zeit, nicht zuletzt wegen der NSA-Affäre und dem Whistleblower Edward Snowden, immer wieder Bestandteil von Diskussionen wenn es um die Übertragung von Kommunikation und Informationen über das Internet geht und ebenfalls ein Faktor für moderne soziale Netze. OpenSource Software gewinnt so natürlich immer mehr an Bedeutung, da offener Quellcode von jedem Menschen eingesehen werden kann und die Wahrscheinlichkeit reduziert, dass eine Applikation seine Nutzer aktiv ausspioniert.

1.2 Aufgabenstellung im Detail

Die rasant fortschreitende Entwicklung im Bereich der Webtechnologien, welche auch immer stärker für die Verwendung auf mobilen Endgeräten wie Smartphones oder Tablets ausgelegt sind, ermöglicht webbasierten Applikationen eine größere Leistungsfähigkeit. Nicht nur durch schnellere Hardware, sondern auch durch bessere Webbrowser und deren JavaScript Interpreter hat ein Webentwickler heutzutage größere Chancen auf verschiedensten Endgeräten, ob Mobil oder Desktop, die gleichen technischen Voraussetzungen vorzufinden und so Applikationen mit einer einheitlichen Codebasis zu schreiben, die auf allen Geräten ohne größere Unterschiede funktionieren.

Diese Arbeit widmet sich dem Ziel, unter Beachtung der zuvor genannten Faktoren, eine experimentelle Basisplattform für sogenannte Social Extranets zu entwickeln. Es soll untersucht werden ob die erarbeitete Lösung für einen Einsatz unter realen Bedingungen geeignet ist und welche Anforderungen erfüllt werden müssen um einen sinnvollen Einsatz der Plattform, auch im Hinblick auf die Erweiterbarkeit und Weiterentwicklung durch andere Entwickler, zu gewährleisten.

Im Kern entstehen so zwei Anforderungen die im Laufe dieser Arbeit bewältigt werden sollen. Die Applikation soll zum einen mit mobilen Geräten benutzbar sein und zum anderen auch noch ohne aktive Internetverbindung (eingeschränkt) funktionieren.

Als wichtigste Komponente für dieses Vorhaben, steht die Datenbank im Mittelpunkt. Im Allgemeinen hat man heutzutage meist noch sehr begrenzte Möglichkeiten um in einem Webbrowser große Mengen an Daten ohne den Einsatz externer Datenbanksoftware direkt im Browser zu speichern. Um die Applikation grundsätzlich unabhängig von externen Einflüssen und darüber hinaus auch offline funktionsfähig zu machen, muss die Datenbank also innerhalb der Laufzeitumgebung (PhoneGap/Webview) bereitgestellt werden.

Die Offline Funktionalität stellt eine weitere Herausforderung da, denn es muss dafür Sorge getragen werden, dass Änderungen welche auf einem Gerät ohne Internetverbindung getätigt wurden, auch auf andere Geräte übertragen werden können, sobald dieses Gerät wieder eine Internetverbindung hat und im Umkehrschluss müssen Änderungen anderer Geräte auch empfangen werden. Das Netzwerk muss also in der Lage sein Datenbanken in alle Richtungen zu replizieren.

1.3 Umfeld

Diese Arbeit entstand im Innovation Lab der Pixelpark AG am Standort Köln. Das Innovation Lab ist eine Forschungsabteilung die aktiv mit neuen und teilweise experimentellen Technologien verschiedenste Projekte umsetzt. Als Teilnehmer des "Future Internet" Programms im Bereich "FIcontent2" der Europäischen Union übernimmt Pixelpark die Verantwortung zur Erstellung eines sogenannten "Social Network Enablers".

Das Social Network Enabler Projekt ist im Bereich der Smart City Services angesiedelt und steht als offene Plattform für Entwickler zur Verfügung, um darauf aufbauend eigene Ideen umzusetzen.

Die in dieser Arbeit erreichten Ergebnisse werden als Kernkomponente in den Social Network Enabler eingebunden.

1.4 Planung

2 Theoretische Grundlagen

2.1 NoSQL

NoSQL ist im allgemeinen ein Begriff für Datenbanksysteme, die nicht dem Prinzip von relationalen Datenbanken folgen. Daten werden nicht relational gespeichert und es wird auch kein SQL als Abfragesprache verwendet.

In der Regel werden bei relationalen Datenbanken sogenannte Transaktionen verwendet, welche bei Zugriff auf einen Datensatz, diesen Datensatz für andere Transaktionen sperren und dabei dem ACID-Prinzip folgen. NoSQL setzt dagegen auf eine andere Vorgehensweise, welche auch als BASE bekannt ist und von Eric Brewer definiert wurde:

- **Basic availability**: Jede Anfrage erhält garantiert eine Antwort
- **Soft state**: Daten können sich auch ohne spezifische Eingaben verändern
- **Eventual consistency**: Die Datenbank kann zeitweise inkonsistent sein, kehrt dann aber zu einem konsistenten Status zurück

In einem verteilten Datenbanksystem definiert Eric Brewer 3 Eigenschaften, welche für verschiedene Anwendungsfälle wichtig sein können: 'Consistency', 'Availability' und 'Partition tolerance'. Das sogenannte CAP-Theorem besagt, das lediglich nur zwei dieser drei Eigenschaften in einem Datenbanksystem garantiert werden können. Im Falle von NoSQL sind Verfügbarkeit und Partitionstoleranz die herausstechenden Merkmale.

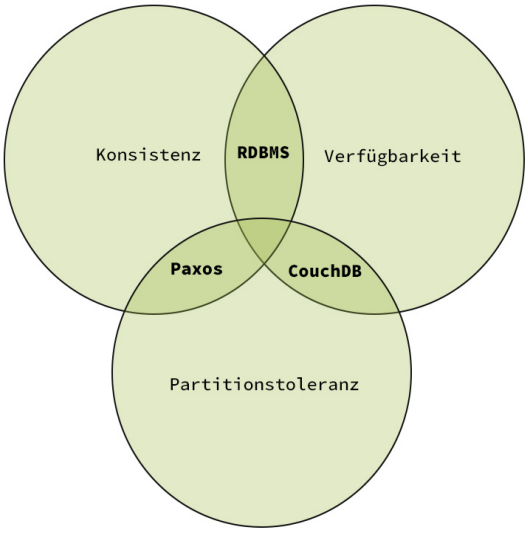

Abbildung 2 - CAP-Theorem

Letztendlich sind die Vorteile von NoSQL Datenbanken vor allem hohe Skalierbarkeit und Geschwindigkeit. Des weiteren ist auch die oft schemalose Repräsentation und die damit verbundene Flexibilität von Datensätzen ein Grund für Entwickler auf NoSQL Datenbanken zu setzen, da hier Entwicklungszeit eingespart wird.

2.2 IndexedDB

Die IndexedDB ist eine HTML5 Spezifikation und befindet sich im Entwicklungsprozess des W3C in der 'Last Call Working Draft' Phase. Die Implementierung in den aktuellen Browsern ist relativ konsistent umgesetzt, d.h. es gibt keine merkbaren Unterschiede. Lediglich Apple mit Safari und iOS und Opera Mini bieten aktuell keine Unterstützung für die IndexedDB-Spezifikation.

IndexedDB macht es möglich große Datenmengen im Browser des Nutzers zu speichern. Die dort abgelegten Daten können mithilfe einer API durchsucht werden.

Im Gegensatz zu Cookies oder Webstorage besteht für die IndexedDB theoretisch kein Limit was die Größe des verfügbaren Datenspeichers betrifft, kann aber durch den Browser begrenzt sein. Jede Applikation die notwendigerweise große Datenmengen übertragen muss, kann davon profitieren diese Daten dauerhaft in dem Browser des Nutzers zu speichern. Ebenso hat der Webstorage keinen Suchmechanismus, sondern nur die Möglichkeit über einen *Key* auf einen *Value* zuzugreifen. Die IndexedDB kommt dem Begriff einer Datenbank also sehr viel näher, als der einfache Webstorage.

2.3 AngularJS

AngularJS ist ein MVC (Model-View-Controller) JavaScript Framework zum Erstellen von interaktiven, dynamischen Single Page Applications.

Bis zum Jahr 2008 war JavaScript eine vergleichsweise langsame Skriptsprache. Durch ständiges Weiterentwickeln der JavaScript Engines, konnte das Ausführen von JavaScript Code in den letzten Jahren jedoch deutlich beschleunigt werden. Durch diesen enormen Performance-Zuwachs in der JavaScript Technologie, konnte man in Erwägung ziehen JavaScript für sogenannte Rich Internet Applications zu verwenden.

Ein Problem das viele JavaScript Frameworks versuchen zu beheben, ist die Tatsache das HTML eine statische Auszeichnungssprache ist. HTML ist nicht dazu konzipiert worden dynamisch zu sein oder dem Nutzer reichhaltige Interaktionsmöglichkeiten zu bieten. Das moderne Web verlangt jedoch nach Lösungen, sowohl im Desktopbereich auch als im mobilen Sektor, Webapplikationen von Grund auf interaktiv zu gestalten.

Eine Methode um den Nutzer mit einer Webapplikation interagieren zu lassen, ist der herkömmliche Weg über einen Webserver, welcher Nutzereingaben verarbeitet und darauf in der Regel eine Antwort an den Nutzer in Form einer HTML Seite ausgibt. Hier entstehen jedoch schon einige Nachteile. Für jede Interaktion des Nutzers muss der Server mit einbezogen werden, was nicht nur Traffic verursacht und vor allem

schlecht für mobile Geräte ist, sondern je nach Auslastung des Servers auch noch längere Ladezeiten verursachen kann.

Eine andere Methode zur Umsetzung interaktiver Webapplikationen ist die Manipulation von HTML und CSS über JavaScript. Als bekanntestes Beispiel ist hier jQuery zu nennen, eine JavaScript Bibliothek die unter anderem eine einfache API zur Manipulation des DOM bietet.

Jedoch ist jQuery keine vollständige Lösung um interaktive Anwendungen zu erstellen, sondern bietet nur eine Schnittstelle zum DOM. AngularJS und vergleichbare Frameworks setzen hier an und bieten zum Teil sehr gut durchdachte Konzepte um Rich Internet Applications effizient zu erstellen.

Eine Fähigkeit von AngularJS ist das 2-way-data-binding, welches den Zugriff auf Models direkt aus der View heraus ermöglicht. Veränderungen am Model können dann durch das Framework erkannt werden und das DOM wird aktualisiert.

```
<body ng-app ng-init="name = 'World'">
    Say hello to: <input type="text" ng-model="name">
    <h1>Hello, {{name}}!</h1>
</body>
```

Abbildung 3 - AngularJS 2-way-data-binding

Das wirklich besondere an AngularJS ist die Fähigkeit HTML um eigene Tags anreichern zu können. Durch die Implementierung sogenannter Direktiven werden Funktionen nicht nur vom restlichen Quellcode abgekapselt und dadurch wiederverwendbar, sondern bekommen in der endgültigen View eine semantische Bedeutung und vereinfachen somit die Lesbarkeit für Entwickler. In Abbildung 4 wird die von AngularJS mitgelieferte Direktive ng-repeat verwendet um über ein Array zu iterieren und dessen Inhalte auszugeben.

```
<ul ng-controller="WorldCtrl">
    <li ng-repeat="country in countries">
        {{country.name}} has population of {{country.population}}
    </li>
    <hr>
    World's population: {{population}} millions
</ul>

<script>
    var WorldCtrl = function ($scope) {
        $scope.population = 7000;
        $scope.countries = [
            {name: 'France', population: 63.1},
            {name: 'United Kingdom', population: 61.8},
        ];
    };
</script>
```

Abbildung 4 - AngularJS ngRepeat Direktive

2.4 Phonegap

Phonegap ist ein OpenSource Framework um die Entwicklung von hybriden Applikationen für Smartphones mittels HTML5, CSS und JavaScript zu ermöglichen.

Eine erste wesentliche Fähigkeit von Phonegap ist das Bereitstellen installierbarer Applikationen für eine Vielzahl von Betriebssystemen, darunter iOS, Android, WindowsPhone, Blackberry, WebOS, Bada und Symbian. Dabei entsteht jede Applikation aus der gleichen Codebasis, bestehend aus HTML, CSS und JavaScript. Um auf allen Betriebssystemen denselben Code ausführen zu können, wird die Applikation in einer Webview ausgeführt.

Eine native Applikation auf dem Handy ist effektiver, wenn man auf gerätespezifische Funktionen, wie zum Beispiel die Kamera, zugreifen kann. Hier stellt Phonegap eine API zur Verfügung die mit JavaScript angesprochen werden kann.

Der Build-Prozess einer Applikation erfordert zusätzlich das SDK des entsprechenden Betriebssystems. Zum Beispiel muss für die Android Kompilierung das Android SDK installiert sein und für die Kompilierung zu einer iOS App wird xCode benötigt.

3 Datenbankreplikation mit CouchDB

Die Entscheidung für ein passendes Datenbanksystem ist für das Ergebnis dieser Arbeit ein sehr wichtiger Faktor. Es gibt zahlreiche Hersteller und damit verbunden eine sehr große Anzahl von entsprechenden Lösungen. Im Folgenden werden die Anforderungen an die Datenbank definiert und die Auswahl für die in dieser Arbeit verwendeten Lösung begründet.

3.1 Analyse der Anforderungen

Die Anforderungen an die Datenbank werden aus einem geplanten Testszenario hergeleitet, welches im Laufe dieser Arbeit an einer Schule mit ca. 15 Testpersonen durchgeführt wird:

"Eine Smartphone Applikation soll für maximal 15 Person ermöglichen, Statusmeldungen in Form von Text über ein Netzwerk in Echtzeit auszutauschen. Das Experiment wird an einem Ort mit mangelhaftem Mobilfunk durchgeführt, sodass Verbindungsabbrüche zum Netzwerk auftreten können."

Aus diesem Szenario ergeben sich folgende Einschränkungen:

- das Netzwerk hat maximal 15 Teilnehmer
- Teilnehmer können Statusmeldungen in Form von Text verschicken
- Statusmeldungen werden nicht editiert
- die Netzwerkverbindung kann unterbrochen werden

Die folgenden drei Anwendungsfälle verdeutlichen das gewünschte Verhalten der finalen Applikation:

3.1.1 Use Case: Initialer Start

```
Vorbedingungen:
App startet zum ersten Mal

Beschreibung:
1. Es wird ein Ladesymbol angezeigt.
2. Alle relevanten Daten aus dem Netzwerk werden zur Offline-Nutzung auf das
Gerät geladen und gespeichert.
3. Die Statusmeldungen werden chronologisch angezeigt.
```

3.1.2 Use Case: Statusmeldung absenden

```
Vorbedingungen:
App ist gestartet; Initialer Start durchgeführt

Beschreibung:
1. Der Teilnehmer gibt seine Statusmeldung ein und drückt auf Senden.
2. Der Teilnehmer ist online, die Statusmeldung wird lokal gespeichert und
über das Netzwerk synchronisiert.

Alternativ:
2b. Der Teilnehmer ist offline, die Statusmeldung wird nur lokal gespeichert.

Nachbedingung:
Die Statusmeldung ist garantiert auf dem Absendergerät gespeichert.
```

3.1.3 Use Case: Statusmeldungen synchronisieren

```
Vorbedingungen:
    Statusmeldungen wurden nur lokal gespeichert; Während der Offline-Nutzung
wurden im Netzwerk neue Statusmeldungen versendet

Beschreibung:
1. Das Teilnehmergerät verbindet sich mit dem Internet
2. Zuvor nur lokal gespeicherte Statusmeldungen werden über das Netzwerk
synchronisiert.
3. Neue Statusmeldungen aus dem Netzwerk werden auf dem Gerät gespeichert und
angezeigt.

Nachbedingung:
Alle neuen Statusmeldungen wurden über das Netzwerk übertragen.
```

3.2 Übertragung der Anforderungen auf die Datenbank

Auf Ebene der Datenbank lassen sich 3 funktionale Anforderungen identifizieren, die benötigt werden um das zuvor beschriebene Szenario erfolgreich bestehen zu können.

A: Kompatibilität zur Laufzeitumgebung PhoneGap (Webview):
Diese Anforderung ergibt sich vorwiegend aus der Wahl von PhoneGap als Zielplattform. Die Datenbank muss mit JavaScript kompatibel sein und über eine entsprechende Schnittstelle verfügen, die auch auf mobilen Geräten verwendet werden kann.

B: Daten können offline gespeichert werden:
Die Datenbank kann Daten auch offline speichern und ist damit unabhängig von einer aktiven Internetverbindung.

C: Synchronisierung zwischen beliebig vielen Geräten:
Um Statusmeldungen auf alle Geräte verteilen zu können, muss die Datenbank in der Lage sein, diese Daten zu synchronisieren.

3.3 Auswahl der Datenbank

Die wohl am schwersten zu bewältigende Anforderung ist durch die Wahl von PhoneGap als Zielplattform entstanden. Die Tatsache, dass die Anwendung in einer Webview laufen wird, verringert die Auswahlmöglichkeiten einer passenden Datenbank.

3.3.1 Lokaler Datenspeicher in PhoneGap

Für HTML5 sind bisher 3 Datenspeicher entwickelt worden:

- Webstorage kann als Ersatz für herkömmliche Cookies betrachtet werden und funktioniert nach dem Key-Value-Prinzip, ist aber eher für kleine Datenmengen ausgelegt.
- WebSQL ist eine Sqlite-Implementation für den Browser, wird jedoch nicht weiterentwickelt und soll daher auch nicht näher betrachtet werden.
- Die IndexedDB ist eine für größere Datenmengen ausgelegte Datenbankschnittstelle zum Speichern von komplex strukturierten Objekten, die durch *Keys* indexiert werden.

Objekte werden in der IndexedDB mit einem *Key* verknüpft. Es gibt keine Query-Language wie bei relationalen Datenbanken üblich, sondern es wird mithilfe eines Cursors über die *Keys* iteriert um Einträge zu finden.

Die API wird über JavaScript angesprochen und Objekte werden in der JSON Notation gespeichert, sind somit also direkt in JavaScript verwendbar. Die IndexedDB wird auch zur Klasse der Dokumentenorientierten Datenbanken gezählt.

```
var request = indexedDB.open("library");

request.onupgradeneeded = function() {
    // The database did not previously exist, so create object stores and
indexes.
    var db = request.result;
    var store = db.createObjectStore("books", {keyPath: "isbn"});
    var titleIndex = store.createIndex("by_title", "title", {unique: true});
    var authorIndex = store.createIndex("by_author", "author");

    // Populate with initial data.
    store.put({title: "Quarry Memories", author: "Fred", isbn: 123456});
    store.put({title: "Water Buffaloes", author: "Fred", isbn: 234567});
    store.put({title: "Bedrock Nights", author: "Barney", isbn: 345678});
};

request.onsuccess = function() {
    db = request.result;
};
```

3.3.2 Synchronisierung

Die lokal gespeicherten Daten in der IndexedDB mit anderen Geräten im Netzwerk zu synchronisieren, ist eine weitere Anforderung, die erfüllt werden muss.

Datenbankreplikationen werden von den meisten modernen Datenbanken unterstützt. Es muss hier aber zwischen zwei Arten von Replikation unterschieden werden. Die meisten Datenbanken müssen früher oder später skaliert werden, um wachsende Schreib- und Lesezugriffe bewältigen zu können. Dafür wird überwiegend die Master-Slave Replikation eingesetzt. Eine im Slave-Modus angebundene Datenbank hat damit die Möglichkeit die Daten der Master-Datenbank zu lesen, besitzt jedoch keine Schreibrechte auf die Master-Datenbank.

Für diese Arbeit ist eine andere Art von Replikation relevant, da die Skalierung bei maximal 15 Teilnehmern keine Rolle spielt. Jedes Gerät im Netzwerk soll Daten auch lokal speichern und später zur zentralen Datenbank replizieren können. Alle Endgeräte besitzen somit eine Master-Datenbank und synchronisieren die Daten zum Server, auf dem sich ebenfalls eine Master-Datenbank befindet. Somit kann jede Datenbank unabhängig von den anderen im Netzwerk befindlichen Datenbanken funktionieren.

Datensätze liegen grundsätzlich als Objekte in der JSON-Notation vor, weshalb eine dazu kompatible Datenbank vorzuziehen ist.

MongoDB und CouchDB sind jeweils Dokumentenorientierte und mit JavaScript kompatible Datenbanken. In Abbildung 5 werden alle relevanten Spezifikationen der Datenbanken gegenübergestellt.

	MongoDB	**CouchDB**
Format	BSON	JSON
Versionierung	Nein	Ja
Map/Reduce	JavaScript + andere	JavaScript
Replikation	Master-Slave	Master-Slave, Master-Master
Protokoll	TCP/IP	HTTP/REST API

Abbildung 5 - Vergleich MongoDB und CouchDB

Die CouchDB übertrumpft im Anwendungsfall dieser Arbeit die MongoDB, da die Unterstützung für JSON-Objekte, Versionierung und Master-Master Replikation gegeben ist.

3.3.3 Gemeinsame API für IndexedDB und CouchDB

Es gibt nicht viele Datenbanken, die eine direkte Kompatibilität zur IndexedDB herstellen. Die PouchDB ist ein im Juni 2010 entstandenes OpenSource Projekt, dass es sich zur Aufgabe gemacht hat, die Funktionsweise der CouchDB direkt in den Browser zu übertragen. Das Projekt befindet sich fortlaufend in Entwicklung, und wurde am 2. Januar 2014 in der Version 1.1.0 veröffentlicht.

Mit der PouchDB ist es möglich, die Replikation zwischen IndexedDB, welche von PouchDB als Speicher verwendet wird, und CouchDB durchzuführen.

3.3.4 IndexedDB, PouchDB und CouchDB

Mit der IndexedDB kommt eine in modernen Browsern verfügbare Datenbank zum Einsatz, welche die Anforderungen A und B erfüllt. Zusätzlich wird die CouchDB in Kombination mit der PouchDB verwendet, um Anforderung C, die Synchronisierung zwischen den Datenbanken, zu erfüllen.

3.4 CouchDB im Detail

Es ist wichtig zu verstehen wie die CouchDB im Detail funktioniert, um einen effizienten Mechanismus zu erarbeiten, der die Daten im Netzwerk zwischen den einzelnen Geräten synchronisieren kann.

Daten werden grundsätzlich als JSON enkodierte Strings gespeichert. Die JSON-Notation ist eine in JavaScript verwendete Schreibweise um Objekte darzustellen. Jedoch wird diese Notation unabhängig von JavaScript, auch von anderen Programmiersprachen verwendet und so hat sich JSON, in Konkurrenz zu XML, zu einem sehr beliebten Format für die Übertragung von Daten innerhalb von Webapplikation entwickelt. Der Vorteil bei Verwendung von JSON als Format in der Applikation dieser Arbeit liegt auf der Hand, denn in AngularJS können die aus der CouchDB gelesenen Daten ohne größeren Aufwand, sprich Umwandlung oder Dekodierung, direkt verwertet werden.

Dokumente in der CouchDB folgen keinem festen Schema. Als Entwickler hat man die Freiheit jederzeit Objekte mit beliebiger Struktur und verschiedenen Datentypen zu speichern. Andere Dokumente werden dadurch nicht beeinflusst.

```
{
    _id: '123',
    _rev: '1-123',
    city: 'Cologne'
}
```

Jedoch gibt es zwei Felder die jedem Dokument zugeordnet werden müssen. Zum einen muss das Feld *_id* einen innerhalb der Datenbank einzigartigen Wert haben und wird verwendet um das Dokument eindeutig zu identifizieren. Der zweite Wert ist das Feld *_rev*, welches die Revisionsnummer speichert. Revisionen entstehen genau dann, wenn ein Dokument verändert wird und in der Datenbank unter derselben *_id* gespeichert werden soll. So lassen sich Änderungen bis zum Erstellen von Dokumenten zurückverfolgen.

Die Abfrage von Daten erfolgt über in der CouchDB erstellte Views und unterscheidet sich damit sehr stark von Datenbankabfragen in relationalen Datenbanksystemen. Jede View enthält eine *Map* Funktion, die auf jedes Dokument einzeln angewendet wird, um einen *key* zu erzeugen.

In der *Map* Funktion kann jeder Wert des aktuellen Dokumentes ausgelesen werden und dazu verwendet werden einen *key* für dieses Dokument zu generieren. Die CouchDB nutzt diese *keys* zur Sortierung in Spalten und macht es so möglich durch Angabe eines bestimmten Bereichs auch große Datenmengen effizient zu durchsuchen und das entsprechende Ergebnis auszugeben.

Das Interessante daran ist, dass die *keys* nicht nur einzelne Werte, sondern auch Arrays sein können. Zueinander in Relation stehende Dokumente können damit in der View hintereinander indexiert werden.

```
{
    _id: 'abc',
    city: 'Cologne'
},
{
    _id: 'def',
    city: 'Zweibruecken'
},
{
    _id: '123',
    place: {
        name: 'Pixelpark AG Cologne',
        related: 'abc'
    }
}
```

In diesem Beispiel wird gezeigt, wie mithilfe einer Map-Funktion *keys* erstellt werden, die der CouchDB eine sortierte Indexierung und Ausgabe ermöglichen.

```
function(doc){
    if(doc.city){
        emit([doc._id, 0], doc.city);
    } else {
        emit([doc.place.related, 1], doc.place.name);
    }
}
```

```
{
    key: ['abc', 0],
    value: Cologne },
{
    key: ['abc', 1],
    value: 'Pixelpark AG Cologne'
},
{
    key: ['def', 0],
    value: 'Zweibruecken'
}
```

Die Replikation von Datenbanken innerhalb eines CouchDB Clusters erfolgt schrittweise. Das bedeutet dass die Datenbanken keine ständige Verbindung zueinander benötigen, wie es bei vielen relationalen Datenbanksystemen der Fall ist. Sobald eine Datenbank einen Replikationsprozess beendet hat ist sie auch eigenständig funktionsfähig.

Die Applikation entscheidet selbst, wann eine Replikation sinnvoll ist und kann diese durch die CouchDB API auslösen. Durch Angabe von Quelle und Ziel wird bestimmt in welche Richtung repliziert wird.

Natürlich kann es passieren dass Datenbanken während einer Replikation Dokumente beinhalten die zueinander in Konflikt stehen. In diesem Fall besitzt die CouchDB einen Algorithmus, der entscheidet welche der beiden Versionen als aktuell gespeichert werden soll. Das Verlierer-Dokument wird aber keinesfalls verworfen, sondern wird unter einer anderen Revision gespeichert. So ist es auch möglich Applikationen zu bauen, die Konflikte nachträglich überprüfen können und gegebenenfalls selbst entscheiden welches Dokument behalten werden soll.

Um Dokumente auf allen Servern zu löschen, benutzt die CouchDB ein *deleted* Feld. Ein Dokument das gelöscht wurde kann keine neuen Revisionen mehr erhalten und wird ebenfalls bei einer Replikation übertragen, sodass auch in allen anderen Datenbanken dieses Dokument als gelöscht markiert wird.

5 Mechanik der Daten-Synchronisierung

Die Applikation benötigt eine Mechanik, welche den Replikationsprozess zwischen der lokalen Datenbank und der CouchDB auslösen kann. Dazu ist es zum einen wichtig die mitgelieferten Methoden der PouchDB zu kennen, zum anderen muss aber auch darauf geachtet werden, dass der erdachte Mechanismus performant ist.

Es ist ebenfalls von Vorteil den Mechanismus so unabhängig wie möglich von der restlichen Applikationslogik zu gestalten. Im Detail bedeutet dies, dass die gesamte Logik zum Schreiben, Lesen und Synchronisieren von Daten als Modul für AngularJS programmiert wird.

5.1 Planung

Die erarbeitete Strategie trennt den Replikationsprozess in 2 Komponenten, das replizieren von der lokalen Datenbank zum Server und umgekehrt. Abbildung 6 zeigt das zugehörige Aktivitätsdiagramm für diese beiden Komponenten.

Abbildung 6 - Aktivitätsdiagramm Synchronisierung vom Server

Die Synchronisierung vom Server zur lokalen Datenbank beinhaltet zwei Schritte und geht davon aus das eine aktive Verbindung zum Netzwerk besteht (ONLINE). Apps die zum ersten Mal gestartet werden haben natürlich noch keine lokale Kopie der Datenbank des Netzwerks. Um den Nutzer aber nicht zu lange auf aktuelle Datensätze warten zu lassen, denn diese würden normalerweise als letztes geladen werden, wird der Initiale Replikationsprozess (bei App-Start) mit einem Filter versehen, welcher nur Dokumente vom Server synchronisiert, die in den letzten 24 Stunden gespeichert wurden. Die Zeitspanne wurde für das geplante Szenario auf 24 Stunden festgelegt, kann für andere Anwendungsfälle jedoch unpassend sein und sollte dann dementsprechend geändert werden.

Nachdem die Initiale Synchronisierung abgeschlossen ist, wird ein weiterer Replikationsprozess gestartet. Diesmal ohne Filter, jedoch als andauernder Prozess, welcher jede Änderung auf dem Server sofort registriert und die entsprechenden Daten auf die lokale Datenbank überträgt.

Durch dieses Vorgehen werden 2 Ziele erreicht, zum einen ein performanter Start für die Applikation, denn es werden zuerst die aktuellsten Daten geladen und zum anderen eine dauerhafte Synchronisierung aller Änderungen auf dem Server.

Abbildung 7 zeigt das Aktivitätsdiagramm zum Replizieren der Daten auf den Server.

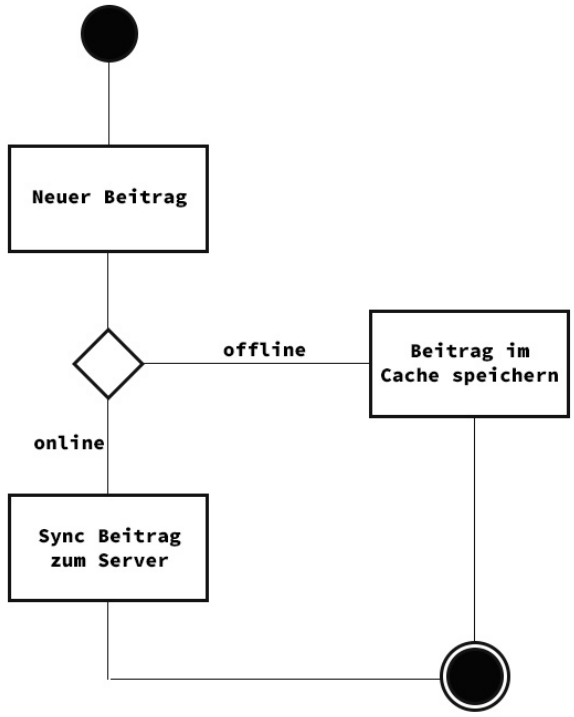

Abbildung 7 - Aktivitätsdiagramm Synchronisierung auf den Server

Die zweite Komponente ist für das Hochladen der Daten auf den Server verantwortlich. Hier macht es wenig Sinn einen dauerhaften Replikationsprozess zu starten, weil neue Daten auf dem lokalen Gerät nur dann entstehen wenn der Nutzer aktiv einen Beitrag hinzufügt. Entgegen dem Vorgehen beim Herunterladen aktueller Daten, wird das Hochladen gezielt beim Speichern neuer Daten ausgelöst.

Nach diesem Vorgehen entsteht für die Offline-Funktionalität ein Hindernis, denn es gibt keine Methode die pauschal alle Daten in der lokalen Datenbank zum Server hoch lädt. Neue Beiträge, die gespeichert werden während das Gerät *Offline* ist, würden so nicht zum Server repliziert werden. Die Lösung hier ist beim Speichern eines neuen Datensatzes direkt zwischen *Online* und *Offline* zu unterscheiden.

Ist das Gerät *Online*, so kann sofort die Methode zum Replizieren auf den Server ausgelöst werden.

Ist das Gerät jedoch *Offline*, wird eine Referenz auf das Dokument zwischengespeichert. Dieser Cache-Mechanismus erlaubt es so alle Änderungen nachträglich zum Server zu replizieren, nämlich genau dann wenn das Gerät wieder eine Netzwerkverbindung erlangt.

Der Nutzen dieses Vorgehens ist eine effektivere Verwendung der Netzwerkapazitäten des Endgerätes. Durch den gezielten Einsatz der Replikation nur auf geänderte bzw. neue Datensätze wird vermieden das alle anderen gespeicherten Daten in der lokalen Datenbank nochmals mit denen der Datenbank auf dem Server abgeglichen werden muss.

5.2 PouchDB API

Zur Umsetzung der Daten-Synchronisierung wird die API der PouchDB verwendet. Die dazu benötigten Methoden werden im Folgenden Abschnitt erläutert.

```
var db = new PouchDB('dbname');
var remote = 'http://couchdb.simple-url.com:5984/dbname';
```

Die Methode zum Erstellen einer Datenbank betrifft die Replikation zwar nicht direkt, jedoch dient es dem besseren Verständnis. Zur Initialisierung der Datenbank wird ein neues PouchDB-Objekt erstellt. Dabei ist es egal ob diese Datenbank schon besteht oder nicht. Beim erstmaligen Aufruf dieses Objekts wird in der IndexedDB die Struktur für die lokale Datenbank erstellt.

Das zurückgegebene PouchDB-Objekt wird in einer Variablen gespeichert und bietet danach Zugriff auf die API der PouchDB.

```
db.post(object, [Options]);
```

Um Dokumente in der lokalen Datenbank zu speichern, wird die Funktion *PouchDB.post* benutzt. Der erste Parameter enthält das zu speichernde Objekt im JSON-Format. Im zweiten Parameter kann ein Objekt mit Optionen übergeben werden. Da fast alle Methoden der PouchDB API asynchrone Funktionsaufrufe sind, werden *Promises* zur Abwicklung der Rückgabewerte verwendet.

Im Abbildung 8 wird ein Dokument gespeichert und der Rückgabewert in der Konsole ausgegeben.

```
db.post({name: 'Hallo FH Zweibruecken'})
    // then wird aufgerufen wenn post fertig ist
    .then(function(response){
        console.log(response);
});
```

Abbildung 8 - Hinzufügen eines Dokuments mit PouchDB

Die Funktion, die letztendlich die Synchronisierung durchführt ist *PouchDB.replicate*. Bei dieser Methode wird im ersten Parameter die URL der entfernten Datenbank übergeben. Der zweite Parameter kann ein Objekt mit Optionen enthalten. Die für diese Arbeit relevanten Optionen sollen im Folgenden erklärt werden.

```
db.replicate.from(remote, [Options]);
db.replicate.to(remote, [Options]);
```

```
db.replicate.from(remote,{
    filter: filterFunction,
    doc_ids: [],
    complete: completeFunction,
    continuous: false
});
```

filter:
Ruft eine Funktion auf um Dokumente selektiv zu replizieren. Hiermit kann erreicht werden, dass nur Daten geladen werden die einen Zeitstempel aufweisen der nicht älter als 24 Stunden ist. Alternativ kann aber auch nur nach Dokumenten gesucht werden die einen bestimmten Wert enthalten.

doc_ids:
Repliziert nur Dokumente mit den angegebenen *ids*.

complete:
Wenn der Replikationsprozess fertig ist, wird diese Funktion aufgerufen.

continuous (seit PouchDB 2.0 *live*):
Diese Option legt fest ob ein Prozess nur einmalig durchlaufen wird und danach beendet wird oder ob der Prozess fortlaufen auf Änderungen wartet.