Fachhochschule Vorarlberg
University of Applied Sciences

Projektdokumentation

Transportüberwachungssystem Roadrunner

Projektteam: Franziskus Domig, BSc; Stefan Gassner, BSc;
Wolfgang Halbeisen, BSc; Matthias Schmid, BSc

Bearbeitung: Dornbirn, im Sommersemester 2011

Betreuer: Prof.(FH) DI Wolfgang Auer

Zusammenfassung

TODO: Hier steht später die Zusammenfassung dieser Arbeit ...

Abstract

TODO: Here is the later to be written abstract of this paper ...

Inhaltsverzeichnis

1 Motivation

Ein Transportüberwachungssystem kann aus mehreren Blickwinkeln betrachtet werden. In diesem Semesterprojekt war für die Erstellung eines Vollständigen Systems zu wenig Zeit vorhanden und somit wurden in diesem Projekt der Fokus auf die Erstellung einer mobilen Applikation, der Repliezrung von Daten auf ein Backendserver sowie die Verwaltung des Systems mit einer Webapplikation gelegt.

Aus erwähnten zeitlichen Gründen wurde keine eigene Hardware entwickelt und somit die Android Plattform als Host-System für eine mobile Applikation verwendet. Zudem wurden bis auf eine Ausnahme keine realen Sensoren verwendet sondern benötigte Sensoren simuliert.

Dadurch hat sich diese Projekt auf die Entwicklung von Software konzentriert. Wir haben uns in neuen Technologien, teilweise sogar in Beta-Versionen, eingearbeitet und diese exzessiv in diesem Projekt verwendet.

Auf dem Backendserver wurde mit CouchDB ein unkonventioneller Ansatz der Datenpersistierung gewählt. Um die Verwaltung von Aufträgen (Lieferungen) wurde eine Webapplikation erstellt.

Als Softwareentwicklungsprozess wurde *Test-Driven-Development* gewählt. Hierzu wurden nahezu alle entwickelten Komponenten mit *Unit-Tests* getestet sowie wenn möglich auf einem *Continius-Integration*-Server bei jeder Änderung automatisiert getestet.

2 Projektanforderungen

Das Ziel dieses Projekts ist es, ein System zur lückenlosen Transportüberwachung zu entwickeln. Hierzu sollen Produkt, welche erst im Rahmen des Projekts zu spezifizieren sind, überwacht werden. Es soll nach einem Transport möglich sein, eindeutig nachvollziehen zu können, welche Sensor-Daten zu jedem Zeitpunkt aufgezeichnet wurde.

Ausgehend von den Anforderungen an ein System für die Transportüberwachung von Arzneimitteln in Österreich [ARGE Pharmazeutika 07] wurde ein System erstellt, welches die Temperatur- sowie die Positionsdaten von Lieferungen bzw. den darin enthaltenen Paketen aufzeichnet und überwacht.

Es soll am Ende dieses Projekts möglich sein, einen einfachen Usecase vollständig durch zu testen.

3 Systembeschreibung und -architektur

In diesem Abschnitt werden die in diesem Projekt verwendeten Technologien erläutert. Insbesondere werden die Gründe beschrieben, weswegen diese Technologien eingesetzt und anderen vorgezogen werden. Die Vor- sowie Nachteile der entsprechenden Technologien werden gegenübergestellt und besprochen. Zugleich werden die entsprechenden Technologien auf ihre Tauglichkeit in einem real logistischen Szenario geprüft.

3.1 Mobiles Gerät auf Basis von Android

Bei diesem Transportüberwachungssystem sollten laut den in Kapitel 2 spezifizierten Anforderungen eine lückenlose und ständige Überwachung sichergestellt werden. Somit müssen Sensorendaten laufend auf ein Backendsystem übertragen werden. Hierzu bietet sich die *Android*[1] Plattform als Grundlage für eine mobile Applikation an.

Mit Android als Grundlage können gleich mehrere Aspekte abgedeckt werden. Jedes Fahrzeug wird mit einem Android Smartphone ausgestattet und ist somit nicht nur telefonisch erreichbar sondern gleichzeitig kann die komplette Transportüberwachung damit erreicht werden.

Mit der entwickelten Applikation lassen sich Gegenstände einladen indem diese via Barcode in das System übernommen werden. Ab diesem Zeitpunkt wird dieser Gegenstand ständig überwacht und via CouchDB (siehe Abschnitt 3.2) mit dem Backensystem synchronisiert. Auf der Webapplikation (siehe Abschnitt 3.3) können die Produkte bzw. die Lieferung, welche aus mehreren Produkten bestehen kann nun auf einer Karte nachverfolgt werden sowie die jeweiligen Daten der Temperatursensoren (siehe Abschnitt 3.4) bis zum ausladen überprüft werden.

Auch in einem realen Szenario lässt sich Android hervorragend einsetzten. Es ist mittlerweile in Version 3.1 (15. Juni 2011) verfügbar und weit verbreitet. Die von Google bereitgestellten *Google Apps for Business*[2] können die Smartphones per Fernwartung administriert werden. Dabei können auch Applikations-Updates an alle registrierten Smartphones verteilt werden. Somit ist auch eine einfache Lösung für das Deployment von neuen Versionen gegeben.

[1] vgl. `http://www.android.com/`

[2] vgl. `http://www.google.com/apps/intl/de/business/index.html`

Im Kapitel 4 wird die in diesem Projekt erstellte Android Applikation beschrieben.

3.2 Verteiltes Datenbansystem CouchDB

Bei einem Transportüberwachungssystem ist die Datensicherung ein wichtiger Aspekt. In diesem Abschnitt wird die Datenverwaltung betrachtet und erläutert welches System für dieses verwendet wird.

In diesem Projekt wurde durch die in Kapitel 2 spezifizierten Anforderungen ein Fokus auf die Verteiltheit des Systems gelegt. Es fallen durch die mobile Transportüberwachung Daten auf mobilen Geräten an, welche mit einem Backendsystem synchronisiert werden müssen. Aus diesem Grund wurde für das Datenbanksystem kein klassisches System in Betracht gezogen. Um einen neuen Ansatz in der Datenpersistierung zu erlernen, wurde das verteilte und dokumentbasierte Datenbankmanagementsystem *CouchDB*[3] verwendet.

In einem real logistischen Szenario muss auf die Skalierbarkeit sowie die Robustheit von CouchDB betrachtet werden. Klassische relationale Datenbankmanagementsysteme bringen durch die bereits sehr gut entwickelten Versionen vor allem einen Vorteil in der Robustheit und Stabilität. Dennoch, CouchDB wird bereits seit 2005 entwickelt und liegt aktuell in der stabilen Version 1.1.0 (6. Juni 2011) vor und wird bereits in mehreren kommerziellen Projekten wie beispielsweise in Ubuntu [Murphy 09][S. 1] eingesetzt.

Einen detaillierten Überblick der Verwendung von CouchDB in diesem Projekt wird in Kapitel 5 gegeben.

3.3 Webapplikation mit dem Framework Silex

Um ein benutzerfreundliches und einfaches Backendsystem für dieses Projekt zu erstellen, wurde auf mehrere bereits bestehende Frameworks zurückgegriffen. *Silex*[4] ist ein Mikroframework für PHP 5.3. Es basiert wiederum auf dem Kern des *Symfony2*[5] Frameworks.

Mit diesem Framework lassen sich einfache Webapplikationen sehr effizient in einer Model-View-Controller Umgebung implementieren. Durch die schöne Tren-

[3] vgl. `http://couchdb.apache.org/`
[4] vgl. `http://silex-project.org`
[5] vgl. `http://symfony.com`

nung der jeweiligen Schichten sowie der leichten Testbarkeit ist Silex für dieses Projekt hervorragend geeignet.

Für ein Szenario in der Realität kann Silex sehr gut eingesetzt werden solange das System einfach und klein ist. Mit mehr in dem Backendsystem implementierten Usecases sollte ein Wechsel zu Symfony2 in Betracht gezogen werden, da sich damit deutlich komplexere Anwendungsfälle implementieren lassen. Ein solcher Wechsel ist durch die bereits in Silex verwendeten Komponenten von Symfony2 leicht zu vollziehen, die bereits bestehenden Komponenten können weiterverwendet werden.

In Kapitel 6 wird eine detaillierte des Backendsystems gegeben.

3.4 Sensoren-Simulation mit Node.js

Für dieses Projekt wurden Temperatursensoren sowie Zeitsynchronisation mit Hilfe von *Node.js*[6] simuliert. Es wurde in diesem Projekt das Augenmerk vor allem auf die mobile Applikation mit Android unter Verwendung einer verteilten Datenbank sowie der Webapplikation gelegt. Somit wurden bis auf Positionssensoren (GPS) die keine richtigen Sensoren verwendet.

Node.js ist ein ereignisgesteuertes I/O Framework für die V8 JavaScript Engine [Wikipedia 10a]. Diese wurde in C++ sowie JavaScript entwickelt und liegt in einer MIT-Lizenz vor, welches es für dieses Projekt einsetzbar macht und zugleich auch in einem realen Szenario eingesetzt werden könnte.

Mit Node.js können mit wenigen Zeilen Code, Server-Applikationen programmiert werden. Es wird hierzu auf einem Interface (IP) sowie einem beliebigen Port eine JavaScript Callback-Funktion registriert, welche bei einem Zugriff aufgerufen wird. Dies macht es sehr einfach, Sensoren in diesem System zu simulieren, welche via HTTP-Requests “ausgelesen” werden können.

In einem real logistischen System werden Sensoren nicht simuliert und somit spielt Node.js nur in diesem simulierten Szenario eine Rolle.

In Kapitel 7 werden die in diesem System mit Node.js simulierten Sensoren erläutert. Zugleich werden die entsprechenden realen Sensoren beschrieben, welche in einer nicht simulierten Umgebung verwendet werden könnten.

[6] vgl. `http://nodejs.org/`

4 Android Applikation

Als Hauptsystem in diesem Projekt wurde eine Applikation für die Android Plattform erstellt. Diese Applikation dient der mobilen Überwachung von Lieferungen bzw. den Gegenständen einer Lieferung.

Android [7] ist ein Betriebssystem sowie auch eine Software Plattform für mobile Geräte. Es werden Smartphones, Netbooks, Mobiltelefone und Tablets unterstützt. Entwickelt wurde das Betriebssystem von der *Open Handset Alliance* [8], einem Konsortium von 80 Firmen zur Schaffung offenen Standards für mobile Geräte, die von *Google* im Jahr 2007 gegründet wurde (vlg. [Open Handset Alliance 07]).

TODO:Was sind Vor- und Nachteile? Was kann man damit alles erreichen? Wo sind die Grenzen?

4.1 Benutzung

TODO

4.2 Implementierung

TODO

4.3 Mögliche Erweiterungen

TODO

4.4 Verwendung in einem realen System

TODO

[7] http://www.android.com/

[8] http://www.openhandsetalliance.com/

5 CouchDB Applikation

Apache CouchDB ist ein Dokument-Orientiertes-Datenbanksystem für die Verwendung mit JavaScript. CouchDB bietet inkrementelle Replikation mit bi-directionaler Konflikt-Erkennung und -Lösung.

CouchDB bietet eine REST-API [Fowler 10][S. 1] via *JavaScript Object Notation* (JSON) an, welche von jeder beliebigen Umgebung mit Hilfe von HTTP-Requests abgerufen werden kann. CouchDB ist zusätzlich ein System, welches eine beliebige Skalierbarkeit sowie Erweiterbarkeit anbietet [CouchDB 11][S. 1].

In diesem Projekt wurde CouchDB eingesetzt, um ein relativ neues Gebiet der Datenpersistierung zu erlernen. Durch die einfache Replizierung von Daten, konnte CouchDB sowohl auf der Backend-Webapplikation (vgl. Kapitel 6) als auch auf den mobilen Geräten (vgl. Kapitel 4) eingesetzt werden.

5.1 Implementierung

TODO: roadrunner.server

5.1.1 JSON und Schema-Validierung

Als Dokumentstruktur wird von CouchDB JSON verwendet. Vor der Speicherung eines JSON-Dokumentes in die Datenbank werden die Validierungsmethoden von allen Designdokumenten in der Datenbank aufgerufen. Nur wenn alle Validierungen erfolgreich sind wird das Dokument gespeichert. Obwohl JSON Schemalos ist kann trotzdem eine Schemavalidierung durchgeführt werden. Als Schema wird das JSON-Schema verwendet, dass sich aktuell in Version 03 befindet [Internet Engineering Task Force 11][S. 1].

In einem Designdokument können verschiedene Validierungen eingeführt werden. Zusätzlich zu Validierungen der Benutzerrechte werden in diesem Projekt alle Dokumente auf das definierte JSON-Schema validiert.

5.2 Datenverteilung

Daten können auf unterschiedliche Arten in einem System verteilt werden. Eine Möglichkeit ist die Daten aus der mobilen Datenbankinstanz in die Applikation zu lesen und auf der Applikationsschicht die anfallenden Daten an die Master-Datenbank zu senden.

Mit CouchDB wird die Synchronisierung der Daten in diesem Projekt von den Datenbankinstanzen durchgeführt. Diese Synchronisierung nennt sich hier Replizierung. Bei dieser kann explizit angegeben werden, welche Daten in jeweils welche Richtung synchronisiert werden.

5.2.1 Designdokumente

TODO

5.2.2 Dokumentänderung - MapReduce

MapReduce[9] ist ein Framework von Google, dass entwickelt wurde damit sehr große Datenmengen parallel bearbeitet werden können. CouchDB verwendet ebenfalls einen ähnlichen Anstaz um Daten aus der Datenbank zu lesen. Anhand eines Beispieles wird die Funktionsweise von MapReduce nachfolgend erläutert.

Map - Phase Auf jedes Dokument in der Datenbank wird die Map-Methode angewendet. In einer Map-Methode werden Key-Value-Paarungen gebildet. Jedes Dokument in der Datenbank kann eine beliebige Anzahl an Key-Value-Paarungen generieren. Diese Key-Value-Paarungen werden in einem B-Baum (vgl. [Ottmann 96][S. 317-327]) gespeichert. Ändert sich nun ein Dokument müssen nur die entsprechenden Paarungen in dem B-Baum angepasst werden.

Reduce - Phase In dieser Phase wird auf jeden Element in dem Baum die Reduce-Methode angewendet. Ziel der Reduce-Methode ist es die Datenmenge zu minimieren. Auf jedes Element kann die Reduce-Methode beliebig oft angewendet werden.

5.3 Mögliche Erweiterungen

TODO

5.4 Verwendung in einem realen Projekt

TODO

[9] vgl. `http://labs.google.com/papers/mapreduce-osdi04.pdf`

6 Webapplikation als Backendsystem

Als unterstützendes Backendsystem wurde in diesem Projekt eine Webapplikation erstellt. Hiermit ist es möglich, eine Lieferung mit entsprechenden Gegenständen zu erstellen. Nach erfolgreichem erstellen einer Lieferung kann diese auf einer Karte nachverfolgt werden. Gleichzeitig kann in einem Diagramm die Temperatur überwacht werden.

6.1 Implementierung

Das Backendsystem wurde in der Programmiersprache *PHP*[10] implementiert. PHP ist eine dynamische Skriptsprache, die speziell für den Einsatz auf Webservern entwickelt wurde. Zum schnelleren Entwickeln, wurden mehrere Frameworks zur Unterstützung verwendet.

6.1.1 Das Silex Framework

Silex[11] ist ein auf *Symfony2*[12] basierendes Mikro-Webapplikations-Framework für PHP 5.3. Es bietet eine überschaubare und intuitive API an, ist einfach zu erweitern und ist durchgängig mit Unit-Tests (PHPUnit[13]) getestet.

In diesem Projekt wurde eine Model-View-Controller [Schmidt 09][S. 354] umgesetzte. Es wurde die Erstellung, Bearbeitung sowie Betrachtung von Lieferungen implementiert. Zusätzlich wurde eine Verwaltung für Transport-Einheiten (z.B. LKWs) und die Benutzerverwaltung für die mobile Applikation in die Webapplikation integriert.

6.1.2 Doctrine2 ODM für CouchDB

Doctrine2[14] ist ein Framework zur Datenbankabstraktion. Im ursprünglichen Framework, war nur ein *Object-Relational-Mapper* (ORM) basierend auf dem *Active-Record-Pattern* [Schmidt 09][S. 380] vorhanden. Dadurch konnten nur relationale Datenbanksysteme (z.B. Oracle oder MySQL) abstrahiert werden. Durch die Ver-

[10] vgl. `http://www.php.net`
[11] vgl. `http://silex-project.org/`
[12] vgl. `http://symfony.com/`
[13] vgl. `http://www.phpunit.de/`
[14] vgl. `http://www.doctrine-project.org/`

wendung einer Dokumentenbasierten-Datenbank (sieht Kapitel 5) in diesem Projekt, wurde allerdings ein *Object-Document-Mapper* (ODM) benötigt.

Dafür hat sich eine Erweiterung von Doctrine2 durch einen ODM für CouchDB angeboten, welcher sich allerdings erst in einem frühen Alpha-Stadion befindet. Dennoch wurde diese Erweiterung verwendet und in einigen Teilen sogar verbessert.

6.1.3 JavaScript Framework jQuery

Das JavaScript Framework *jQuery*[15] ist eine schnelle und einfach zu bedienende Bibliothek um HTML-Dokument Manipulationen, Ereignis-Behandlung, Animierung sowie Effekte und Ajax- Interaktionen durchzuführen.

jQuery wurde entwickelt und schneller Webapplikationen zu entwickeln. Es wurde der Fokus vor allem auf die Art wie JavaScript in Webapplikationen verwendet wird gelegt.

In diesem Projekt wurde die Darstellung von Graphen, Karten sowie die Validierung von Benutzereingaben mit jQuery realisiert.

6.1.4 Blueprint CSS Framework

Für die Erstellung eines einfachen *Grid-Layout* [W3C 11][S. 1] mithilfe von *Cascading-Style-Sheets* (CSS) wurde in diesem Projekt das CSS-Framework *Blueprint*[16] verwendet.

Hiermit lässt sich schnell ein grobes Grundgerüst für moderne Webapplikationen erstellen. Es verwendet eine ansprechende Typographie und bietet dem Designer bzw. Entwickler einen guten Ansatz für die Layoutgestaltung. Bei Bedarf kann dieses Framework mit einigen Plugins erweitert werden.

6.2 Mögliche Erweiterungen

Aus unternehmerischer Sicht wäre eine Anbindung an eine Kundendatenbank von großer Bedeutung. Somit können Lieferungen an wiederkehrende Auftraggeber einfacher eingetragen werden.

Eine andere wichtige Erweiterungsmöglichkeit, wäre die Implementierung einer entsprechenden Zugriffskontrolle. Ein entsprechendes Rechtesystem für diese Webapplikation wäre vor allem in einem realen System von großer Bedeutung.

[15] vgl. `http://jquery.com/`

[16] vgl. `http://www.blueprintcss.org/`

6.3 Verwendung in einem realen System

In einem realen System ist möglicherweise bereits eine Backend-Applikation vorhanden, welche um die entsprechenden Komponenten erweitert werden müsste. Die Backend-Datenbank (siehe Kapitel 5) kann auch an eine andere Applikation angebunden werden. Beispielsweise könnte eine Integration in eine SAP[17] Umgebung erfolgen.

Sollten keine eigene Backend-Applikation im Unternehmen bestehen, sollte diese Webapplikation entsprechend erweitert werden. Beispielsweise muss ein Zugriffskontrolle, eine Kundendatenbank etc. implementiert werden.

[17] vgl. `http://www.sap.com/germany/index.epx`

7 Transportüberwachung mittels Sensoren

In diesem Projekt wurden die Transportüberwachung mittels Sensoren, welche von der Android Applikation (siehe Kapitel 4)) überwacht werden, realisiert.

Für die in diesem Projekt spezifizierten Anforderungen (siehe Kapitel 2) wurde die Temperatur sowie die aktuelle Position eines Gegenstands überwacht. Hierzu werden zwei unterschiedliche Sensortypen, welche in den beiden nachfolgenden Abschnitten erläutert werden, verwendet. Zusätzlich wurde die Zeitsynchronisation der mobilen Geräte mittels eigens entwickelter Synchronisierung (wie in Abschnitt 8.1 erläutert) realisiert.

7.1 Temperaturüberwachung

Temperatursensoren werden in diesem Projekt simuliert. Alle benötigten Temperatursensoren werden mit *nodejs*, wie in Abschnitt 3.4 erläutert, simuliert.

7.2 Positionsüberwachung

8 Applikationssicherheit

8.1 Zeitsynchronisierung

In diesem Abschnitt werden Probleme besprochen, die durch fehlerhafte respektive mangelhaft durchdachte Zeitsynchronisierung oder Verbindungsabbruch entstehen können.

Problem durch falsche Zeitstempel bei Logeinträgen: Betrachtet wird das Szenario "Umladevorgang eines Produktes". Das mobile Gerät der Transporteinheit wird benutzt um den Ausladevorgang aus einem Container im System zu verarbeiten. Mit dem scannen des Produkts wird auf dem mobilen Gerät der Transporteinheit ein Logeintrag in dessen lokale Datenbank erstellt. Genauso wird beim darauffolgenden Ladevorgang der Umladestation ein Logeintrag auf dessen Gerät erstellt. Wenn das System mit absoluter Zeit arbeitet und die Uhrzeit des Geräts der Transporteinheit vor jener der Umladestation ist, dann würde im System der Übernahmevorgang der Umladestation vor dem Ausladevorgang der Transporteinheit stattfinden.

Lösungsansatz: Um dieses Problem zu lösen muss relative Zeit eingeführt und synchronisiert werden. Für die Zeitsynchronisierung können bekannte Algorithmen für verteilte Systeme verwendet werden. Mögliche Algorithmen sind Christian's Algorithm oder Berkley Algorithm[18].

Im Projekt Roadrunner wurde Christian's Algorithmus implementiert. Jeder Client misst seine Differenz zur Serverzeit und verwendet diese zur Erzeugung der Zeitstempel. Somit können die unterschiedlichen Uhren bis zu einer gewünschten Genauigkeit synchronisiert werden.

Grundsätzlicher Ablauf zur Ermittlung der Zeitdifferenz ist folgender:

1. Client erzeugt einen Zeitstempel mit der letzten bekannten Differenz zur Serverzeit.
2. Client sendet eine Anfrage für die Serverzeit an den Server.
3. Server antwortet mit seiner Zeit.
4. Client erzeugt einen zweiten Zeitstempel.

[18] $http://en.wikipedia.org/wiki/Clock_synchronization$

5. Ist die RoundTrip-Zeit klein genug werden die Zeitstempel von Client und Server verglichen.

6. Überschreitet die Zeitdifferenz einen Schwellwert wird auf dem Client die neue Zeitdifferenz gesetzt.

Die Uhren werden bis zu einer gewählten Genauigkeit miteinander synchronisiert. Beim Projekt Roadrunner wurde die Genauigkeit mit 5 Sekunden gewählt. Dies bedeutet, wenn der Client seinen Zeitstempel mit dem von dem Server vergleicht und die Differenz die Genauigkeit überschreitet, dann ermittelt der Client die neue Differenz und verwendet diese zur Erzeugung der nächsten Zeitstempel. Die Synchronisation bis zu einer Genauigkeit von 5 Sekunden wurde gewählt, da ein Umladevorgang sicher länger als 5 Sekunden dauert. Eine zu kleine Genauigkeit hätte zur Folge, dass durch Abweichende RoundTrip-Zeiten ständig die Uhr neu gestellt würde und unnötig viele Zeitsynchronisierungseinträge erstellt würden.

Bei jeder Zeitsynchronisierung wird eine Log-Eintrag für alle geladenene Items erzeugt. Dadurch ist in der History eines Items ersichtlich wann eine Zeitsynchronisierung des Client durchgeführt wurde und welche Log-Einträge einen von der Serverzeit abweichenden Zeitstempel haben.

Bei der Zeitsynchronisierung ist die RoundTrip-Zeit ebenfalls von großer Bedeutung. Ist die RoundTrip-Zeit zu groß wird der Zeitstempel vom Server verworfen und keine Synchronisierung durchgeführt. Nur wenn die RoundTrip-Zeit hinreichend klein ist, kann eine sinnvolle Zeitdifferenz der Zeitstempel ermittelt werden. Für die größte noch erlaubte RoundTrip-Zeit wurde 2 Sekunden gewählt. Dies entspricht etwa der halben Genauigkeit und bei keiner zu großen Netzauslastung sollte ein Request in dieser Zeit auch abarbeitbar sein.

8.2 Zugriffskontrolle

Zur Umsetzung des Rechtesystems von Roadrunner werden verschiedene Benutzergruppen eingeführt. Die Rechte werden einerseits direkt auf der Datenbank definiert und zudem noch über Validierungsfunktionen umgesetzt. Die Benutzerauthentifizierung wird von CouchDB durchgeführt.

8.3 Administratoren & Benutzer

Auf einer Datenbank können in CouchDB Admins und Readers definiert werden.

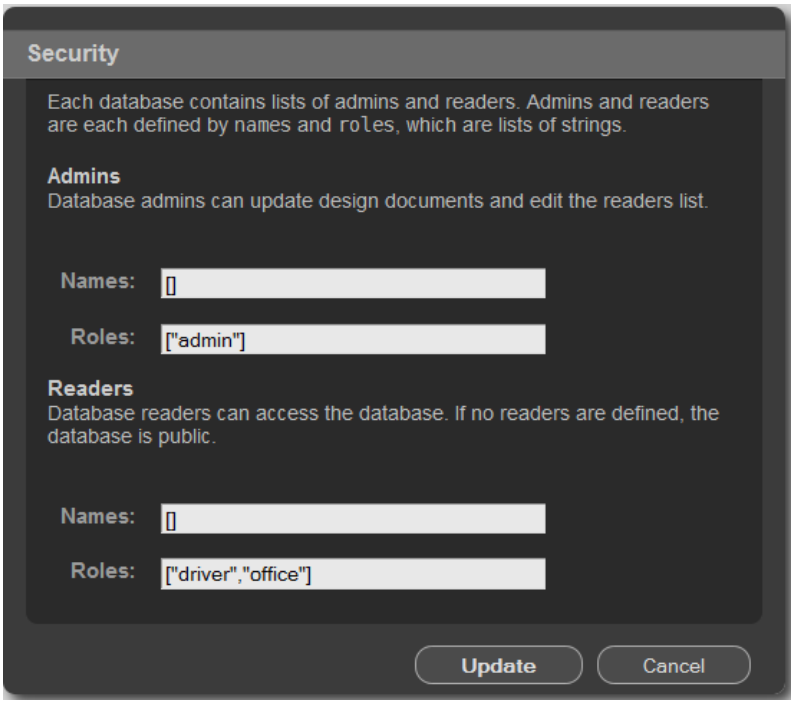

Abbildung 8.1: Admins & Readers

Admin Ein Admin hat sämtliche Rechte auf der Datenbank. Er kann die zudem die Datenbank löschen, Designdokumente verändern oder Benutzerrechte ändern.

Reader Ein Reader hat lesenden und schreibenden Zugriff auf alle Dokumente bis auf die Designdokumente.

Ein Benutzer kann über 2 verschiedene Arten einer Gruppe zugeordnet werden:

- Names: Ein CouchDB-Benutzer muss einen eindeutigen Namen im Format „org.couchdb.user:[username]“ haben. Die Benutzer werden in einer seperaten Datenbank mit dem Namen „_users“ definiert. Wenn der Benutzer dieser Liste (Array aus Strings) hinzugefügt wird, dann besitzt er die entsprechenden Rechte.

- Roles: Ein Benutzer kann verschiedene Rollen besitzen. Wenn eine seiner Rollen in dieser Liste aufgeführt wird, hat er die entsprechenden Rechte.

8.3.1 Rollen in Roadrunner

Einem Benutzer können keine bis mehrere Rollen zugewiesen werden. Bei jeder Veränderung von Dokumenten auf dem Backendsystem wird von CouchDB eine Benutzerauthentifizierung durchgeführt. Bei dieser Benutzerauthentifizierung wird

	Driver	Office	Admin
Benutzerrechte ändern	-	-	+
Designdokumente ändern	-	-	+
Logeinträge anlegen	+	+	+
Logeinträge ändern	-	-	+
Logeinträge löschen	-	-	+
Andere Dokumente anlegen	-	+	+
Andere Dokumente ändern	-	+	+
Andere Dokumente löschen	-	+	+

Tabelle 8.1: Benutzerrechte

das Zugriffsrecht auf die Datenbank überprüft und zudem eine Validierung durchgeführt. Bei der Validierung werden alle definierten Validierungsmethoden aufgerufen. Nur wenn alle Validierungen gültig sind wird die gewünschte Änderung an den Dokumenten durchgeführt.

Im Projekt Roadrunner wurden 3 verschiedenen Rollen definiert:

Admin Ein Admin hat sämtliche Rechte auf der Datenbank. Diese Rollen ist ausschließlich für Administratoren vorgesehen.

Office Ein Benutzer der Gruppe Office arbeitet mit dem Backendsystem von Roadrunner. Dieser Benutzer arbeitet über die Webapplikation mit Roadrunner.

Driver Ein Benutzer der Gruppe Driver ist ein Fahrer. Er arbeitet auf dem Androidsystem mit Roadrunner. Auf dem Androidsystem arbeitet er als Admin mit der Datenbank. Eine Einschränkung der Benutzerrechte auf dem Androidsystem ist nicht nötig da die Benutzerauthentifizierung bei der Replizierung der Daten von dem Androidsystem auf das Backendsystem durchgeführt wird. Ein Fahrer kann nur Daten replizieren für die er die entsprechenden Rechte besitzt.

In den Validierungsmethoden werden die entpsrechenden Rechtevalidierungen durchgeführt. In Tabelle 8.1 sind die Berechtigungen aufgelistet. Ein + bedeutet, dass der Benutzer das Recht besitzt.

9 Wirtschaftliche Betrachtung

TODO

10 Projektentwicklung

In diesem Projekt wurde ein agiler Softwareentwicklungsansatz gewählt. Es wurde die Entwicklung von Software in den Vordergrund gestellt und der klassischen, formalisierten Vorgehensweise geringe Bedeutung zugeteilt. Diese Vorgehensweise kann mit der von [Beck 98][S. 25] et al. entwickelten Methode *Extreme-Programming* verglichen werden.

Um diesen, durch fortlaufende Iterationen und den Einsatz mehrerer Einzelmethoden, sich stets ändernden Prozess erfolgreich anwenden zu können, ist eine entsprechende Disziplin sowie Kommunikationsbereitschaft im Team notwendig.

Durch den sogenannten *Best-Practice* Ansatz kann in dieser Art der Entwicklung auf bereits vorhandene Lösungsansätze zurückgegriffen werden und somit der Entwicklungsprozess erheblich beschleunigt werden.

Da die jeweiligen Iterationen nur kleine Änderungen und in der Regel nur ein neues Feature in das System einführen, können auch sehr einfach neue Technologien getestet werden. Sollte es sich herausstellen, dass eine neue Technologie nicht die erwarteten Verbesserungen bringt, kann durch diese Vorgehensweise auch wieder schnell auf eine vorherige Iteration zurück gewechselt werden.

Konkret wurde in diesem Projekt in abwechselnden zweier Teams entwickelt und jedes neue Feature bzw. neuer Quellcode durch beide Entwickler besprochen und bei Bedarf verbessert.

Mit der Unterstützung eines verteilten Versionskontrollsystem (siehe Abschnitt 11.1) konnte auch sehr einfach gleichzeitig an mehreren Features gearbeitet werden.

11 Projektunterstützende Werkzeuge und Hilfsmittel

In diesem Projekt wurden mehrere Projektunterstützende Werkzeuge verwendet. Durch den sehr agilen Entwicklungsprozess (siehe Kapitel 10) wurden vor allem auf eine entsprechende Versionskontrolle sowie ein Continious-Integration-Server zur Überwachung des jeweiligen Entwicklungsstands geachtet. Diese beiden Systeme sind in den nachfolgenden Abschnitten erläutert.

11.1 Versionskontrollsystem GIT mit github.com

TODO

11.2 Continious Integration mit Jenkins

TODO

12 Zusammenfassung

TODO

Literatur

[Beck 98] K. Beck, W. Cunningham, R. Jeffries: Chrysler Goes To “Extremes”, Distributed Systems: Case Study, S. 25-28 October 1998, online abrufbar: http://www.xprogramming.com/publications/dc9810cs.pdf.

[Gamma 94] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, MA: Addison-Wesley, 1994.

[Ottmann 96] T. Ottmann, P. Widmayer: Algorithmen und Datenstrukturen-3, Heidelberg; Berlin; Oxford: Spektrum, Akad. Verlag 1996.

[Schmidt 09] S. Schmidt: PHP Design Patterns: Entwurfsmuster für die Praxis., Köln: O’Reilly Verlag, 2. Auflage 2009.

Web-Referenzen

[ARGE Pharmazeutika 07] Arbeitsgemeinschaft des pharmazeutischen Großhandels Österreichs: Codex für den Transport von Arzneimitteln in Österreich http://www.argepgh.at/kwpc_Downloads/Pharmarecht%20%D6sterreich/Codex%20Transport%201007.pdf, besucht am 15.03.2011.

[CouchDB 11] Apache CouchDB: Technical Overview http://couchdb.apache.org/docs/overview.html, besucht am 22.06.2011.

[Fowler 10] M. Fowler: Richardson Maturity Model: Steps towards the glory of REST http://martinfowler.com/articles/richardsonMaturityModel.html, besucht am 20.06.2011.

[Internet Engineering Task Force 11] Internet Engineering Task Force: A JSON Media Type for Describing the Structure and Meaning of JSON Documents http://tools.ietf.org/html/draft-zyp-json-schema-03, besucht am 11.05.2011.

[Murphy 09] E. Murphy: CouchDB in Ubuntu http://mail-archives.apache.org/mod_mbox/couchdb-dev/200910.mbox/%3C4AD53996.3090104@canonical.com%3E, besucht am 16.06.2011.

[W3C 11] W3C Working Draft 7 April 2011: Grid Layout http://www.w3.org/TR/css3-grid-layout/, besucht am 22.06.2011.

[Wikipedia 10a] Wikipedia: Node.js http://de.wikipedia.org/wiki/Node.js, besucht am 20.04.2011.

[Open Handset Alliance 07] Open Handset Alliance http://www.openhandsetalliance.com/press_110507.html, besucht am 23.06.2011.

A Appendix

A.1 Kommerzielle Temperatursensoren

Hygrosens TLOG20-BLUE Das ist *NICHT* unsere Lösung. http://shop.hygrosens.com/Messsysteme-acma/Messsysteme-fuer-Temperatur/Temperaturmesssysteme/Temperaturmesssysteme-BLUETOOTH/BLUETOOTH-Temperaturmesssystem-20-Kanaele.html hygrosens.com/TLOG20-BLUE, Zugriff am 16.04.2011

Ampedrf BT11 Das ist *NICHT* unsere Lösung. http://www.ampedrf.com/modules.htm BT11 Class1, Zugriff am 16.04.2011 http://www.ampedrf.com/datasheets/BT11_Datasheet.pdf BT11 Datasheet

$149 Programmable Universal Key Fob Sensor Wir haben uns für das BlueRadios BR-FOB-SEN-LE4.0 Device entschieden, weil es eine komplette und etablierte Lösung für Temperatur, Beschleunigungs- und Licht-Messung ist. http://www.blueradios.com/BR-FOB-SEN-LE4.0-S2A.pdf Blueradios BR-FOB-SEN-LE4, Zugriff am 16.04.2011

http://www.blueradios.com/hardware_sensors.htm Blueradios BR-FOB-SEN-LE4

A.1.1 Datenbankinstanzen

Anfallende Daten müssen persistent gespeichert werden. Diese Speicherung wird in eine Datenbank durchgeführt. Genauer betrachtet werden die Daten in einer Instanz einer Datenbank gespeichert. Es gilt zu unterscheiden ob eine Instanz einer Datenbank verwendet wird oder mehrere Instanzen verwendet werden und diese synchron gehalten werden.

Eine Instanz Bei der Verwendung einer Datenbankinstanz gibt es einen zentralen Datenbankserver. Alle Daten werden von dieser Instanz gelesen und geschrieben. Der Vorteil dabei ist, dass alle gespeicherten Daten auf dieser Instanz sofort zur Verfügung stehen. Der Nachteil ist, dass die Datenbank für alle Clients durchgehend zur Verfügung stehen muss.

Mehrere Instanzen - Jeder synchronisiert mit jedem Wenn mehrere Instanzen einer Datenbank verwendet werden gilt es diese synchron zu halten. Dies bedeutet, wenn auf einer Instanz Daten erzeugt werden, müssen diese mit anderen Instanzen synchronisiert werden. Eine Möglichkeit ist, dass jede Instanz mit allen anderen Instanzen synchronisiert wird. Dies wäre eine Lösung, wenn es eine definierte Menge von Instanzen gibt und jede Instanz von jeder Instanz aus erreicht werden kann. Dadurch ergibt sich Fehlertoleranz gegenüber Ausfällen von einzelnen Datenbankinstanzen, da die Daten auf jeder Instanz vorliegen und es keine zentrale Masterinstanz gibt.

Mehrere Instanzen - Jeder synchronisiert mit der Masterinstanz Anstatt dass jede Instanz mit jeder anderen Instanz synchronisiert wird kann auch eine zentrale Masterinstanz verwendet werden. Diese zentrale Instanz hält alle Daten und verteilt diese Daten wenn nötig auf andere Instanzen. Diese bedeutet wenn eine Clientinstanz neue Daten generiert hat werden diese zur Masterinstanz gesendet und wenn der Client bestimmte Daten benötigt kann er diese bei der Masterinstanz abholen. Dadurch ergibt sich aber ein zentraler Fehlerpunkt. Wenn die Masterinstanz ausfällt ist keine Datenverteilung mehr möglich.

Da beim Roadrunner-Projekt die Daten auf mobilen Geräten erzeugt werden und diese oft auch offline arbeiten, müssen die Daten auf dem Gerät ebenfalls gespeichert werden. Diese Daten werden auf dem Gerät in einer Datenbankinstanz gespeichert. Da es einem mobilen Gerät nicht möglich ist zu allen anderen mobilen Geräten im System Kontakt aufzunehmen wird eine zentrale Masterinstanz verwendet. Auf ein mobiles Gerät werden nur solche Daten gespeichert, die für die Abwicklung der Lieferaufträge benötigt werden.

A.1.2 Datenspeicherung

Daten können in einer Datenbank auf unterschiedliche Arten gespeichert werden. Dieser Abschnitt beschreibt die unterschiedlichen Speicherungsarten und beschreibt ob diese für das Projekt Roadrunner verwendet werden können.

Relationale Datenbank In einer relationalen Datenbank werden Daten in einer Tabelle gespeichert. Tabellen werden über PrimaryKey-ForeignKey-Verknüpfungen miteinander in Verbindung gebracht. Ein Eintrag in eine

Tabelle, die auf eine andere Tabelle verweist, kann nur durchgeführt werden wenn der Eintrag auf den verweisen wird in der anderen Tabelle existiert. Somit müssten sehr viele Daten auf die mobilen Geräte verteilt werden. Ein Beispiel: Ein Temperatur-Log-Eintrag gehört zu einem Sensor und zu einem Warengut. Das Warengut befindet sich in einem Transportbehälter und muss somit mit diesem Verknüpft werden. Der Transportbehälter hat ein Fahrzeug. Ein Fahrzeug gehört zu einem Fuhrpark usw. Die Verwendung einer relationalen Datenbank auf einem mobilen Gerät wäre nur möglich wenn unterschiedliche Datenbankschemas für die Instanzen der mobilen Geräte und des Masters verwendet werden.

Objektorientierte Datenbank Die Daten werden direkt als Objekte in die Datenbank gespeichert. Da auf den mobilen Geräten aber Java-Objekte bestehen und auf der Webapplikation PHP-Objekte verwendet werden ist diese Lösung nicht ohne intensiven Programmieraufwand für die Konvertierung möglich.

Dokumentdatenbank Die Daten werden als Dokumente in die Datenbank gespeichert. Ein Log-Eintrag ist ein Beispiel solch eines Dokumentes. Dokumente sind für sich unabhängige Datensätze die beliebig in einem System verteilt werden können. Dokumente haben Versionsnummern. Anhand der Versionsnummer erkennt eine Datenbankinstanz ob es eine alte Version eines Dokumentes besitzt und kann bei der Masterinstanz eine neue Version des Dokumentes abholen.

Bei Roadrunner wird eine verteilte Dokumentdatenbank verwendet. Die verwendete Datenbank verwendet als Dokumentstruktur JSON. Für JSON gibt es eine hohe Integration in Java und PHP und ist somit eine einfach zu verwendende Datenstruktur.

A.2 Usecases

A.2.1 Login

Ein Fahrer hat im System einen Benutzer. Durch Benutzername und Passwort kann sich der Fahrer auf dem Android-System einloggen. Als Authentifizierungssystem zum Server werden dabei CouchDB-User verwendet. Details zum Authentifizierungssystem sind unter Abschnitt 8.

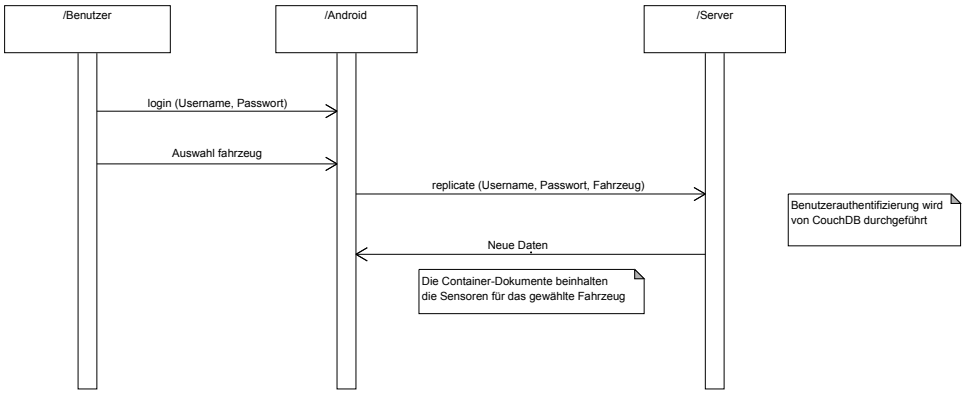

Abbildung A.1: Loginvorgang

Nach dem Einloggen kann der Fahrer sein Fahrzeug wählen. Das Auswählen des Fahrzeuges ist notwendig, damit das Android-System weiß welche Sensoren auszulesen sind. In Kapitel 7 ist beschrieben wie jedes Fahrzeug mit Sensoren ausgestattet wird.

In Abbildung A.1 ist ersichtlich wie der Loginvorgang durchgeführt wird. Nach dem Loginvorgang versucht das System eine Anfrage für neue Daten an den Server zu senden. Bei einer aktiven Serververbindung werden die Benutzerdaten und das gewählte Fahrzeug an den Server gesendet. Nach einer Authentifizierung des Benutzers wird ermittelt ob für das gewählte Fahrzeug neue Daten bezüglich der Sensoren vorhanden sind. Sind neue Daten vorhanden werden diese per Datenbankreplikation an das mobile Gerät gesendet.

Bei der Replikation werden Container-Dokumente übertragen. In diesen Dokumenten sind die Informationen gespeichert, welche Sensoren auf dem jeweiligen Fahrzeug vorhanden sind und wie diese anzusprechen sind.

- Wareneingang

  registriert Pakete im System

  klebt QR-Code auf Pakete

- Logister plant und erstellt Lieferungen (neue Auftragsnummer wird generiert)

  wählt Pakete aus (können mit Sensoren bestückt sein)

  wählt Fahrer aus

  wählt Transportmittel (können mit Sensoren bestückt sein) für Lieferungen aus

trägt Zielort und Auftraggeber ein

- Transporteur

  holt oder hat Device mit Roadrunner App

  loggt sich im Roadrunner System ein

  mit Benutzerdaten wird sein/e aktuelle/r Auftrag/Lieferung aufs Device synchronisiert ODER

  scannt Pakete und lädt sie in das vom Logistiker ausgwählte Transportmittel

Daten-Synchronisierung Vorbedingungen:

- Transporteur hat sich in der System-App eingeloggt

Die Daten-Synchronisierung oder Replizierung wird durch das Einloggen im System angestoßen. Das mobile Gerät erhält folgende Information:

- Adressen der Sensoren, die das Gerät überwachen sollte
- alle Produkte, Pakete der aktuellen Lieferung, sowie Zielort, etc.
- Überwachungs-Thresholds der Pakete