FACHHOCHSCHULE VORARLBERG GMBH

Dokumentation

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

Inhaltsverzeichnis

1 Vorwort

TODO

2 Spezifikation

Roadrunner ist ein System zur Überwachung von Arzneimittel-Transporten. Es verwaltet Waren in einem Backend-System, welches ein verteiltes Datenbanksystem verwendet. Mit mobilen Android-Geräten wird der Transport sowie die Lagerung von Zustellern sowie Lagermitarbeitern überwacht.

Transport von medizinischen Produkten (temperatursensitiv, Transportzeit, überwachbar, etc.).

2.1 Technologie

- Backend-/Application-Server
- Annahmestelle für Waren (Item)
- verteilte Datenbank (CouchDB)
- Android für mobile Überwachung der Sensoren/Position/etc.
- Sensoren
  - WLAN: Zugriff mit bspw. `http://192.168.47.11:1234/`
  - Bluetooth: TODO Matze
- Barcode-Lesegerät mit Android Devices
  - Barcode-Scanner App

    ```
    Activity.forResult (com.google.zxing.client.android.SCAN);
    ```

2.2 Fragestellungen

CouchDB für diese Anwendung sinnvoll, machbar? Ja.

Barcode mit Android-Device lesbar? Schnell genug? Ist ein Framework verfügbar?

Sensoren mit Bluetooth auslesbar? PAN? Alternativen (evt. WLAN/HTTP)?

3 Technologie

In diesem Abschnitt werden die von 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 CouchDB

CouchDB[1] ist eine auf die Verwaltung von Dokumenten basierte Datenbank. CouchDB kann ähnlich wie das MapReduce Framework von Google[2] abgefragt sowie indiziert werden.

TODO: Ausführlicher beschreiben.

3.2 Node.js

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.

3.3 PHP - Silex

PHP ist eine dynamische Skriptsprache, die speziell für den Einsatz auf Webservern entwickelt wurde. Jegliche Anfrage an eine PHP-Datei auf einem entsprechend konfigurierten Webserver wird durch die *PHP-Runtime* interpretiert. Dabei wird eine passende Antwort generiert und durch den Webserver an den Client gesendet. Dies

[1] The Apache CouchDB Project, `http://couchdb.apache.org/`

[2] vgl. `http://en.wikipedia.org/wiki/MapReduce`

stellt die Basis für eine dynamische Webseite mit PHP dar. PHP ist verfügbar für eine breite Anzahl an Webservern, sowie für unterschiedlichste Plattformen wie Linux, Mac OS X, Windows oder Unix.

Silex[3] ist ein Mikroframework für PHP 5.3. Es basiert wiederum auf dem Kern des Symfony2[4] Framework.

3.4 Android/Java

TODO

3.5 Barscanner

TODO

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

4 Datenbank

Bei einem Überwachungssystem für Warentransporte werden sehr viele Daten entstehen. Diese Daten gilt es zu Verwalten und deren Integrität sicherzustellen. In diesem Kapitel werden unterschiedliche Möglichkeiten der Datenverwaltung betrachtet und erläutert welches System für das Projekt Roadrunner verwendet wird.

4.1 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.

4.2 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.

4.3 Datenverteilung

Daten können auf unterschiedliche Arten im System verteilt werden. Eine Möglichkeit wäre die Daten aus der mobilen Datenbankinstanz in die Applikation zu lesen und das Senden der Daten an die Masterinstanz über die Applikation durchzuführen. Eine andere Möglichkeit ist es, die Datenbanksynchronisierung direkt von den Datenbanken durchführen zu lassen.

Bei der verwendeten Datenbank im Roadrunner-Projekt wird die Synchronisierung der Daten von den Datenbankinstanzen durchgeführt. Diese Synchronisierung nennt sich Replizierung und es kann dabei angegeben werden welche Daten synchronisiert werden sollen.

4.4 CouchDB

Dieser Abschnitt beschreibt wie CouchDB die Anforderungen erfüllt.

4.5 Alternative Datenbanksysteme

Dieser Abschnitt beschreibt die möglichen alternativen Datenbanksysteme (z.B. Cassandra) und warum CouchDB als Datenbanksystem ausgewählt wurde.

5 CouchDB

5.1 JSON

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.[5]

5.1.1 Schema-Validierung

In einem Designdokument können verschiedene Validierungen eingeführt werden. Zusätzlich zu Validierungen der Benutzerrechte werden im Projekt Roadrunner alle Dokumente auf das definierte JSON-Schema validiert. Zur Durchführung der Schema-Validierung wurde ein spezielles Designdokument erzeugt. In diesem Designdokument befinden sich folgende Elemente:

validate_doc_update.js Die JavaScript-Methode `function (newDoc, oldDoc, userCtx )` wird vor der Speicherung von jedem Dokument aufgerufen. In dieser Methode wird die Schema-Validierung durchgeführt. Die Methode besitzt keinen Rückgabewert. Wenn die Methode ohne aulösen einer Exception durchläuft gilt das Dokument für diese Validierungsmethode als valide.

schema Die verwendeten Schemas befinden sich in diesem Ordner im JSON-Format

vendor/json-schema Zur Durchführung der Validierung wird das Projekt json-schema[6] verwendet. Bei dem Projekt handelt es sich um eine kleine JavaScript-Klasse, die keine zusätzlichen Abhängigkeiten hat und alle benötigten Funktionalitäten zur Verfügung stellt.

Ein alternatives Validierungsframework wäre das JSON-Schema-Validator-Projekt[7]. Das JSV-Framework bietet die selben Funktionalitäten wie das json-schema-Projekt. JSV verwendet aber Aufrufe an die Java-Script-Methode

[5] http://tools.ietf.org/html/draft-zyp-json-schema-03

[6] https://github.com/kriszyp/json-schema

[7] https://github.com/garycourt/JSV.git

`require(...)`. Diese Methode wurde von der CouchDB-Validierungsmethode überschrieben und dient dort zur Validierung einzelner Elemente in einem JSON-Dokument. Durch das Überschreiben der Methode von CouchDB arbeitet der `require`-Befehl im JSV-Framework nicht korrekt und somit kann dieses nicht verwendet werden.

5.2 Dokumentstruktur

5.3 Designdokumente

5.4 Replizierung

5.5 MapReduce

MapReduce ist ein Framework von Google, dass entwickelt wurde damit sehr große Datenmengen parallell bearbeitet werden können. CouchDB verwendet ebenfalls einen MapReduce-Anstaz um Daten aus der Datenbank zu lesen. Anhand eines Beispieles wird die Funktionsweise von MapReduce vorgestellt.

Das Beispiel beantwortet folgende Problemstellung: Welche Waren wurden gescannt und somit als geladen gekennzeichnet?

5.5.1 Map - Phase

Auf jedes Document in der Datenbank wird die Map-Methode angewendet. In einer Map-Methode werden Key-Value-Paarungen gebildet. Jedes Document in der Datenbank kann eine beliebige Anzahl an Key-Value-Paarungen generieren. Diese Key-Value-Paarungen werden in einem B-Tree gespeichert. Ändert sich nun ein Dokument müssen nur die entsprechenden Paarungen in dem B-Tree angepasst werden.

5.5.2 Reduce - Phase

In der Reduce-Phase wird auf jeden Node in dem Tree die Reduce-Methode angewendet. Ziel der Reduce-Methode ist es die Datenmenge zu minimieren. Auf jede Node kann die Reduce-Methode beliebig oft angewendet werden. Daher wird die Reduce und die Rereduce-Phase unterschieden.

5.6 CouchDB auf Android

5.7 Sicherheit

5.8 CouchApp

6 Android

6.1 Voraussetzungen

6.2 Setup der Applikation

6.3 Komponenten

Dieser Abschnitt beschreibt aus welchen Komponenten die Applikation besteht. Es gibt einen Service und eine Main-Activity.

7 GIT

7.1 Unterschiede zu SVN

7.2 Arbeitsweise mit GIT

7.3 Mögliche Probleme

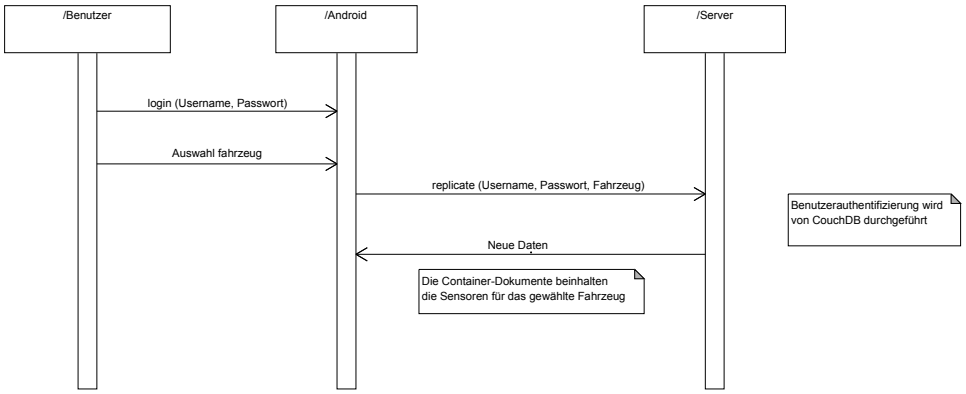

Abbildung 8.1: Loginvorgang

8 Usecases

8.1 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 5.7.

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 11 ist beschrieben wie jedes Fahrzeug mit Sensoren ausgestattet wird.

In Abbildung 8.1 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

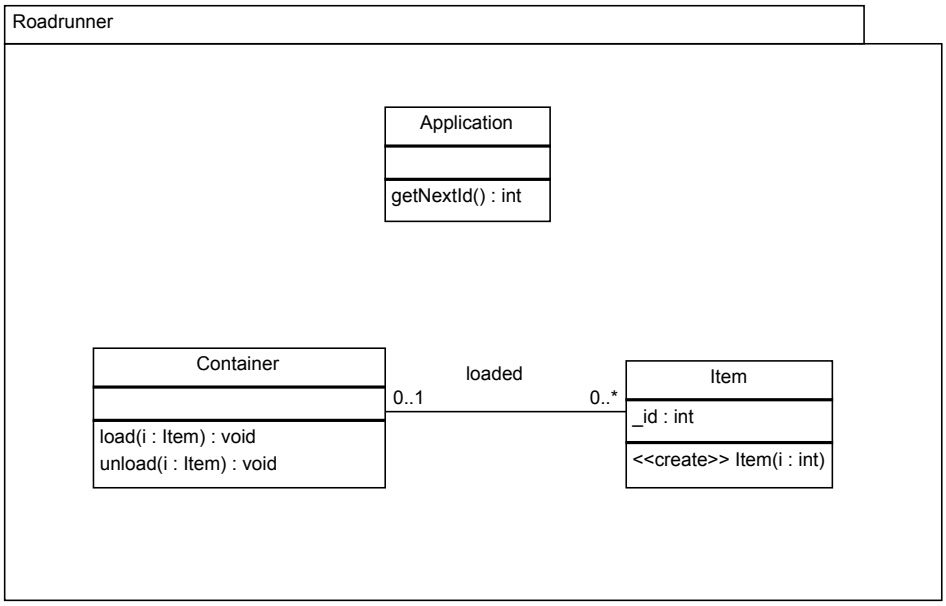

Abbildung 9.1: Iteration 1

9 Iteration 1

9.1 Ziele

- Produkte können erzeugt werden.
- Produkte können eingelagert werden.
- Produkte können ausgelagert werden.

9.2 UML

10 Sicherheit

10.1 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 eingeführt werden. Mögliche Algorithmen sind

TODO: UPV distributed Clocks .. algorithmen herausfinden und oben einfügen

Christian's Algorithm, Berkley Algorithm, $http://en.wikipedia.org/wiki/Clock_synchronization$

Grundsätzlich müssen diese Probleme berücksichtigt werden. In unserem Projekt werden die erwähnten Lösungen aus zeitlichen Gründen und anderer Zielsetzung nicht implementiert.

11 Sensorik

11.1 Typen

11.2 Wann und wie kommt der Sensor ins System?

11.3 Überwachung

11.4 Wie werden dem Mobilen Device Sensoren zugeordnet?

11.5 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

11.6 Simulation

Wir haben uns in diesem Projekt entschieden, die Thematik Sensor vollständig zu simulieren. Einerseits aus zeitlichen Aspekten und andererseits, um unseren Hauptfokus intensiver ausarbeiten zu können.

11.7 Wie sieht unsere Sensorsimulation aus?

Alle benötigten Sensoren, Zeitsensoren sowie Temperatursensoren werden mit *node-js*, wie in Abschnitt 3.2 erläutert, simuliert.

Literatur

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

Web-Referenzen

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