Bachelorarbeit

Möglichkeiten des Einsatzes von NoSQL-Technologien im Umfeld von Krankenhausinformationsystemen

Andy Krebs

Geboren am 15. Mai 1990 in Werdau

Studiengang Informatik

Westsächsische Hochschule Zwickau

Fakultät Physikalische Technik / Informatik

Fachgruppe Informatik

Betreuer:	Prof. Dr. Thomas Franke
Abgabetermin:	16. August 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Gegenstand und Motivation

Gegenstand der Bachelorthesis sind Einsatzmöglichkeiten von NoSQL-Technologien im Umfeld von Krankenhausinformationssystemen (KIS). Die Ausgabe des Themas erfolgte durch Prof. Dr. Thomas Franke, welcher an der Westsächsischen Hochschule Zwickau tätig ist.

Ein weiterer wichtiger Gegenstand der Arbeit ist CouchDB, welcher einer der bekanntesten Vertreter von NoSQL-Technologien ist und zu dem Gebiet der dokumentorientierten Datenbanken gehört. Mit Hilfe von CouchDB soll in dieser Arbeit herausgefunden werden, inwieweit man im Umfeld von Krankenhausinformationssystemen relationale Datenbanken durch dokumentorientierte Datenbanken ersetzen könnte. Dementsprechend wird davon ausgegangen, dass die Datenspeicherung momentan mit Hilfe einer relationalen Datenbank erfolgt. Es soll ermittelt werden, welche Unterschiede zwischen RDBMS und Dokument orientierten Datenbanken existieren. Durch geeignete Anwendungsbeispiele soll festgestellt werden, ob der Einsatz von NoSQL-Technologien im Umfeld von KIS Sinn macht.

1.2 Problemstellung

1.2.1 Unterschiede RDBMS und Document Stores

Es ist nicht hinreichend bekannt, welche Unterschiede zwischen relationalen Datenbankmanagementsystemen und dokumentorientierten Datenbanken bestehen.

1.2.2 Analyse der Einsatzmöglichkeiten

Es ist unklar, wie NoSQL-Technologien sinnvoll im Umfeld von Krankenhausinformationssystemen zum Einsatz kommen können.

1.3 Problemstellung

1.3.1 Ziel zu Problem 1.2.1

Im Rahmen dieser Arbeit werden die Unterschiede zwischen RDBMS und Document Stores identifiziert und vorgestellt.

1.3.2 Ziel zu Problem 1.2.2

Geeignete Implementierungen sollen mögliche Einsatzgebiete im Umfeld von KIS hervorbringen.

2 Grundlagen

In diesem Kapitel werden grob die Grundlagen für relationale Datenbanken beschrieben. Dafür ist es wichtig zu wissen, was ein relationales Datenmanagementsystem ist und was dessen wichtigste Eigenschaften aussagen. Außerdem wird definiert, was ein Krankenhausinformationssystem ist.

2.1 Krankenhausinformationssystem (KIS)

Der Begriff Krankenhausinformationssystem steht für die Gesamtheit aller informationsverarbeitenden Systeme im Krankenhaus, welche zur Erfassung, Bearbeitung und Weitergabe von medizinischen und administrativen Daten verwendet werden. Ein KIS besteht meist aus mehreren Anwendungsbausteinen, die miteinander kommunizieren können. [Wik13c]

2.2 Relationales Datenbankmanagementsystem

Ein relationales Datenbankmanagementsystem ist ein Datenbankmanagementsystem, welches relationale Datenbanken beinhaltet. Die Daten werden in einzelnen Tabellen gespeichert und können in Beziehung miteinander stehen. Aufgrund dessen werden

komplexe Datenobjekte in viele einfachere Datenobjekte zerlegt. Mit Hilfe von Relationen lassen sich Beziehungen zwischen den Datenobjekten realisieren. Die meisten relationalen Datenbanken benutzen dafür die Datenbanksprache SQL. [Wik13a]

2.3 ACID – Prinzip

Damit in einem Datenbanksystem sinnvoll Transaktionen durchgeführt werden können, muss das System die ACID - Eigenschaften garantieren. Eine Transaktion ist eine Abfolge von bestimmten Programmierschritten, die eine logische Einheit bilden. Im Einzelnen steht ACID für Atomarität (atomicity), Konsistenz (consistency), Isolation (isolation) und Dauerhaftigkeit (durability). Atomarität bedeutet, dass eine Transaktion entweder ganz oder gar nicht ausgeführt wird. Im Falle eines Abbruchs bleibt das System unverändert. Bei Konsistenz spricht man von einem konsistenten Datenbestand nach Ausführen einer Transaktion. Wenn mehrere Transaktionen parallel ausgeführt werden, dient die Isolation dazu, dass sie sich nicht gegenseitig beeinflussen. Das Ergebnis einer Transaktion muss außerdem dauerhaft bestehen bleiben. Das bedeutet, dass die Transaktionen dauerhaft gesichert sind. [Wik13b]

3 NoSQL

NoSQL steht nicht, wie man vermutlich auf dem ersten Blick denkt, für „no“ SQL. Es beschreibt im Wesentlichen viel mehr eine Bewegung, welche im Sinne von „not only“ SQL denkt. Ziel ist es nicht gegen SQL zu sein, sondern Alternativen zu herkömmlichen Lösungen zu finden. In diesem Sinne spricht man also höchstens von Alternativen zu SQL. An dieser Stelle wird eine mögliche Definition aus dem NoSQL-Archiv in deutscher Übersetzung zitiert:

> „Definition:
>
> Unter NoSQL wird eine neue Generation von Datenbanksystemen verstanden, die meistens einige der nachfolgenden Punkte berücksichtigen:
>
> 1. Das zugrundeliegende Datenmodell ist nicht relational.
> 2. Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.

3. Das NoSQL-System ist Open Source.
4. Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.
5. Aufgrund der verteilten Architektur unterstützt das System eine einfache Datenreplikation.
6. Das System bietet eine einfache API.
7. Dem System liegt meistens auch ein anderes Konsistenzmodell zugrunde: Eventually Consistent und BASE, aber nicht ACID.“ [Han11]

In diesem Abschnitt wird noch nicht genauer auf die Definition und Begriffe eingegangen, da dies in Kapitel 4 CouchDB ausführlicher an einem typischen Vertreter erfolgt.

3.1 Geschichte

Ken Thompsen entwickelte bereits 1979 die erste Key/Value-Datenbank namens DBM. Aus den 80er Jahren entstanden weitere Vorreiter wie Lotus Notes, BerkleyDB und GT.M, welche noch heute sehr beliebt sind. Der Begriff NoSQL wurde das erste Mal von Carlo Strozzi im Jahre 1998 verwendet. Seine Datenbank hatte keine SQL-API, jedoch lag immer noch ein relationales Datenbankmodell zugrunde.

Mit dem Jahr 2000, Web 2.0 und dem Versuch große Datenmengen zu verarbeiten, kam der Vorstoß von NoSQL. So kann man Google als den NoSQL-Vorreiter schlechthin bezeichnen. Mit der Entwicklung grundlegender Technologien wie MapReduce, BigTable-Datenbanksystem und dem eigenen Hochgeschwindigkeits-Dateisystem legten sie den Grundstein. Natürlich zogen anschließend weitere Internetgrößen wie Yahoo, Amazon, MySpace oder auch Facebook nach.

In dem Zeitraum von 2006 bis 2009 entstanden die heutigen klassischen Vertreter von NoSQL-Systemen wie Cassandra, CouchDB, Dynamo/Dynamite, HBase, Hypertable, Neo4j, MongoDB, Redis, Riak und viele mehr. Seit 2009 wurde NoSQL dann das erste Mal im Sinne von „Not Only SQL“ bezeichnet, wodurch die Systeme mehr in die öffentliche Wahrnehmung rückten. [Han11,FHK13]

3.2 Kategorisierung von NoSQL – Systemen

Im Laufe der NoSQL - Bewegung haben sich vier wichtige Kategorien herauskristallisiert, welche im Folgenden beschrieben werden.

3.2.1 Key/Value - Systeme

Wie der Name schon vermuten lässt, können diese Systeme einen Value besitzen, welcher einem Schlüssel zugeordnet ist. Die Schlüssel können meistens in Datenbanken und Namensräume aufgeteilt werden. Die Values müssen auch nicht nur aus Zeichenketten bestehen. Sie können genauso gut Hashes, Sets oder Listen sein.

Der klare Vorteil dieser Systeme ist die Skalierbarkeit, sowie das einfache Datenmodell, wodurch man eine schnellere und effizientere Datenverwaltung hat. Dadurch lassen allerdings die Abfragemöglichkeiten zu wünschen übrig, da der Zugriff nur über den Schlüssel erfolgt.

Typische Vertreter in diesem Bereich sind DynamoDB, Redis, Voldemort und Scalaris. [Han11]

3.2.2 Column-Family-Systeme

Column-Family-Systeme sind spaltenorientiert und ähneln den Datenstrukturen von Excel-Tabellen. Besonders ist dabei, dass beliebige Schlüssel auf beliebig viele Key/Value-Paare angewendet werden können. Jede Spalte kann außerdem mit beliebig vielen Key/Value-Paaren erweitert werden. Daher kommt auch der Name Column Family. Wenn mehrere Column Families unter einem Keyspace gegliedert werden, spricht man in diesem Fall von Super-Columns. [Han11]

3.2.3 Graphendatenbanken

Unter Graphendatenbanken versteht man die Verwaltung von Graph- beziehungsweise Baumstrukturen. Die Elemente darin sind miteinander verknüpft und im Falle von soge-

nannten nativen Graphendatenbanken kann man die Kanten und Knoten der Graphen mit Eigenschaften versehen.

Zum Beispiel kann man sagen „Person (Kind) spielt Karten (Mau Mau)“. Diesen Ausdruck kann man als kleinen Graph verstehen. So würde man in einer Graphendatenbank drei Einträge („Person“, „Relation“, „Karten“) anlegen und diese mit den jeweiligen Properties („Kind“, „spielt“, „Mau Mau“) versehen. [Han11]

3.2.4 Document Stores

Dokumentorientierte Datenbanken (oder Document Stores) zählen zu einer der wichtigsten Kategorien. Das wichtigste Merkmal ist hierbei, dass die Daten in einem sogenannten Dokument gespeichert werden. Unter einem Dokument ist eine Zusammenstellung bestimmter Daten zu verstehen. Dies können zum Beispiel JSON, YAML oder RDF-Dokumente sein. So legt eine dokumentorientierte Datenbank beispielsweise JSON-Dateien mit einer ID ab.

Ein Beispiel für ein JSON-Dokument könnte wie folgt aussehen:

```
{
 „Vorname“: „Max“,
„Nachname“: „Mustermann“,
„Alter“: „25“
}
```

Zu erwähnen gibt es noch, dass die Dokumente schemafrei und horizontal skalierbar sind. Zu den wichtigsten Vertretern zählen CouchDB, MongoDB und Riak. [Han11]

4 CouchDB

„Apache CouchDB has started. Time to relax.“ [Han11]

Diese Aussage soll den Grundgedanken von CouchDB wiederspiegeln. Man setzt auf eine einfache Bedienbarkeit, wodurch lediglich Wissen über HTTP und JavaScript benötigt wird. Die Datenbanken interagieren dabei über eine HTTP-RESTful-Schnittstelle mittels JavaScript.

Desweiteren rechnet CouchDB damit, dass nicht immer eine Netzwerkverbindung vorhanden ist. Daher kommt auch der Name „Cluster of unreliable commodity Data Base", was zu Deutsch heißt: „Datenbank an einem Cluster aus unzuverlässiger Standardhardware". Außerdem weist CouchDB ein großes Plus in Sachen Performance und Skalierbarkeit auf.

Im Jahre 2005 begann die Entwicklung von CouchDB auf privater Grundlage von Damien Katz. Er schrieb seine Anwendung zunächst in C++, wechselte im April 2008 allerdings auf die Sprache Erlang, da er den Wunsch hatte, Fehlertoleranz zu erreichen. Im selben Jahr trat er bis 2009 der Apache Software Foundation bei. Somit wurde CouchDB offiziell anerkannt und zum vollwertigen Apache-Projekt ernannt.

CouchDB gehört zu der Kategorie der dokumentorientierten Datenbanken von NoSQL. Damit steht schon mal fest, dass CouchDB dokumentbasiert und somit schemafrei ist. Außer Attachments werden alle Daten ausschließlich im JSON-Format gespeichert. Weitere Informationen zu CouchDB werden im Folgenden aufgelistet und später noch näher betrachtet. [Cou11,Han11]

- Webadresse: http://couchdb.org
- Abfragen werden über MapReduce ausgeführt
- API: RESTful JSON API
- Zuverlässige Integrität durch MVCC
- Speichert BLOBS als Attachment im Dokument
- CouchDB ist OpenSource
- Inkrementelle Multi-Master-Replikation mit Konflikterkennung
- Skalierung erfolgt über Replikation
- Geschrieben in Erlang (Programmiersprache)
- Protokoll: http (Portnummer 5984)
- Integriertes Webinterface Futon (http://127.0.0.1:5984/_utils/)
- In der Version 1.3.1 verfügbar (Stand: 02.07.2013)

4.1 Anatomie der Dokumente

Die wichtigsten Merkmale eines Dokuments sind die ID und Revisionsnummer. Sie dienen der eindeutigen Identifikation und zeigen die aktuelle Revision des Dokuments. Welche Rolle

die beiden Merkmale genau spielen, wird in den folgenden Kapiteln ersichtlich. Ein weiteres wichtiges Merkmal ist, dass die Dokumente im JSON-Format geschrieben werden, was bereits erwähnt wurde. Die folgende Abbildung ist eine Dokumentansicht im Webinterface Futon. Was die Vorteile von Dokumenten sind, wird im Laufe der Arbeit genauer erläutert und hier vorerst nicht weiter erwähnt.

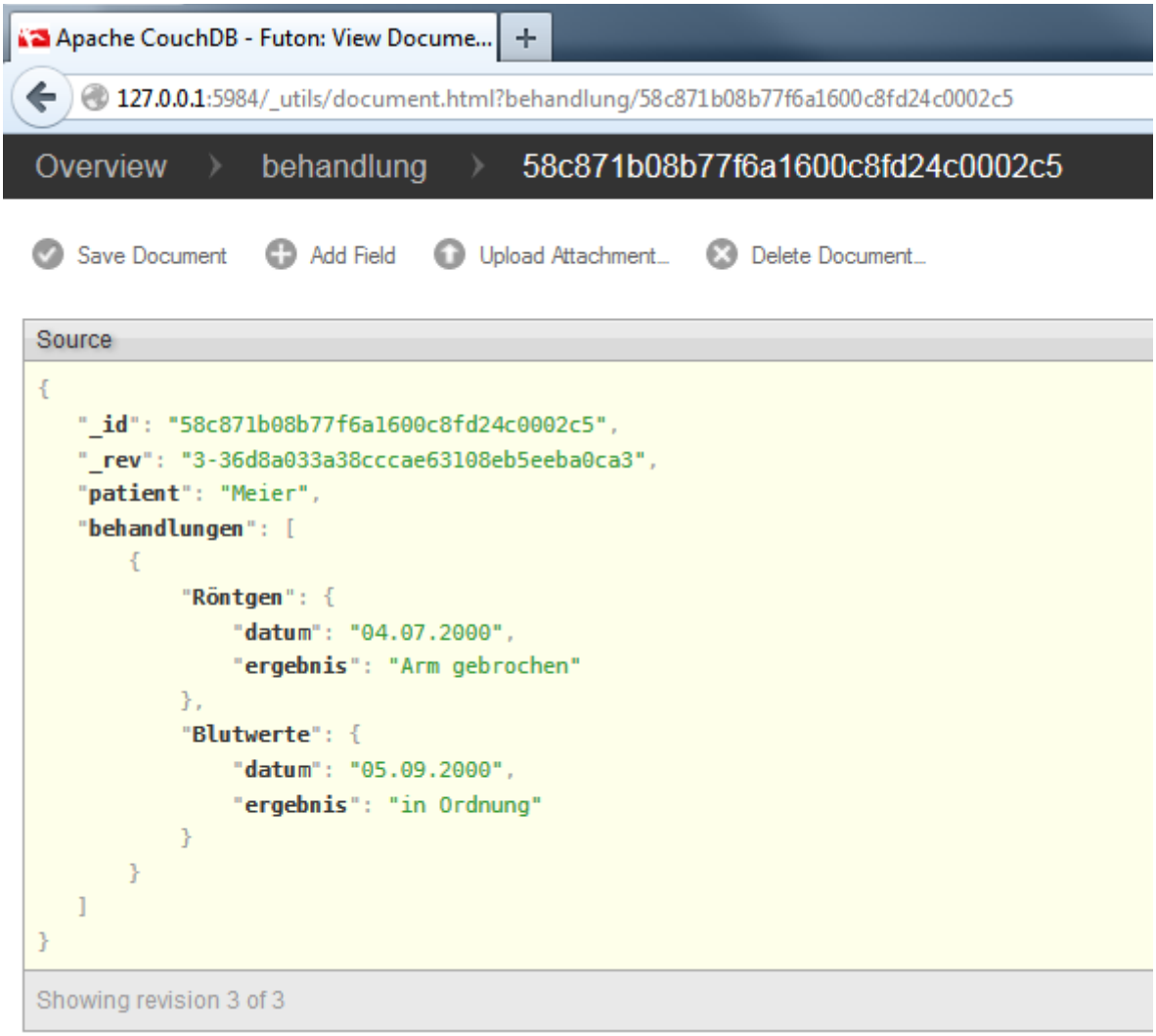

Abbildung 1 Dokumentansicht im JSON-Format in Futon

4.2 RESTful

RESTful wurde im Jahre 2000 von Roy Thompson eingeführt und dient dem Zugriff auf Ressourcen über HTTP-Methoden. REST steht hierbei für Representational State Transfer, welches ein Programmierstil für Webanwendungen ist. RESTful besitzt im Wesentlichen drei wichtige Eigenschaften. Zum einen brauch es einen Uniform Resource Identifier (URI). Dieser beschreibt eine Ressource oder auch eine Menge von Ressourcen. An diesem URI kann nun

ein HTTP-Request erfolgen. Wichtig hierbei ist allerdings dass der Request die Informationen mitbringen muss und keinen State (Zustand), auf dem Server beispielsweise, voraussetzen darf. Dies wäre also die zweite wichtige Eigenschaft von RESTful, der Zugriff auf Ressourcen muss immer zustandslos sein. Und zu guter Letzt brauch man natürlich noch die Information darüber, was mit der Ressource geschehen soll. Durch den Einsatz von HTTP-Methoden, wie PUT und GET, lässt sich dies beschreiben [Cou11]. In der folgenden Tabelle werden die einzelnen HTTP-Methoden und ihre Funktionen aufgezeigt:

HTTP-Methode	Beschreibung
OPTIONS	Liefert Informationen über die verfügbaren Kommunikationsoptionen für den aktuellen URI.
GET	Liefert die Repräsentation der Ressource, die durch den URI identifiziert wird.
HEAD	Liefert die Metainformationen wie der GET-Request, allerdings ohne Message-Body
POST	Legt eine untergeordnete Ressource an dem angegebenen URI an.
PUT	Modifiziert oder legt eine neue Ressource an dem angegebenen URI an.
DELETE	Löscht die Ressource an angegebenen URI.
TRACE	Kann genutzt werden, um zu kontrollieren, mit welchen Veränderungen z.B. der Webserver den Request empfangen hat.
CONNECT	Ist für die Nutzung mit einem Proxy und die Erstellung von (z.B. SSL-) Tunnels reserviert.
COPY	Ermöglicht das Kopieren eines Dokuments in CouchDB (nicht HTTP-Standard)

Tabelle 1 HTTP-Methoden [Cou11]

Das Ausführen von Operationen in CouchDB erfolgt ausschließlich mit Hilfe dieser HTTP-Methoden. Um ein besseres Verständnis davon zu bekommen, soll folgendes Beispiel helfen. Wichtig hierfür ist zunächst das Kommandozeilen-Programm cURL und eine installierte CouchDB-Instanz. CURL ist eine Bibliothek, welche Befehle in einer Shell oder Eingabe-aufforderung ausführt und über die HTTP-Schnittstelle von CouchDB, HTTP-Methoden senden kann. In diesem Beispiel wird zunächst eine neue Datenbank, über die HTTP-Methode PUT, mit dem Namen testcouch angelegt:

```
c:\>curl.exe -X PUT 127.0.0.1:5984/testcouch
{"ok":true}
```
Abbildung 2 HTTP-Methode PUT

Mit den folgenden Befehlen lässt sich nun mittels GET-Request überprüfen, ob die neu angelegte Datenbank auf der Instanz vorhanden ist, oder man lässt sich gleich alle Datenbanken anzeigen:

```
c:\>curl.exe 127.0.0.1:5984/testcouch
{"db_name":"testcouch","doc_count":0,"doc_del_count":0,"update_seq":0,"purge_seq
":0,"compact_running":false,"disk_size":79,"data_size":0,"instance_start_time":"
1375295910437000","disk_format_version":6,"committed_update_seq":0}

c:\>curl.exe 127.0.0.1:5984/_all_dbs
["_replicator","_users","behandlung","mapreduce","testcouch"]
```
Abbildung 3 HTTP-Methode GET

Anschließend wird die neu angelegte Datenbank per DELETE wieder gelöscht:

```
c:\>curl.exe -X DELETE 127.0.0.1:5984/testcouch
{"ok":true}
```
Abbildung 4 HTTP-Methode DELETE

4.3 Futon

Im Umfang des Installationspaketes von CouchDB ist Futon enthalten. Futon ist eine Web-Applikation und bietet den vollen Umfang aller Funktionen von CouchDB (Abbildung 5). Die Applikation ist auf dem lokalen Rechner über *http://127.0.0.1:5984/_utils/* zu erreichen. Darin kann man seine Datenbanken und die einzelnen Dokumente anlegen, anzeigen, bearbeiten oder löschen. Weiterhin lassen sich auch sämtliche Funktionen, die beispielsweise eine Anwendung betreffen, damit bearbeiten. Kurz gesagt, man kann über diese leicht bedienbare Web-Applikation alles erledigen und muss nicht nur über die Kommandozeile arbeiten. [Cou11]

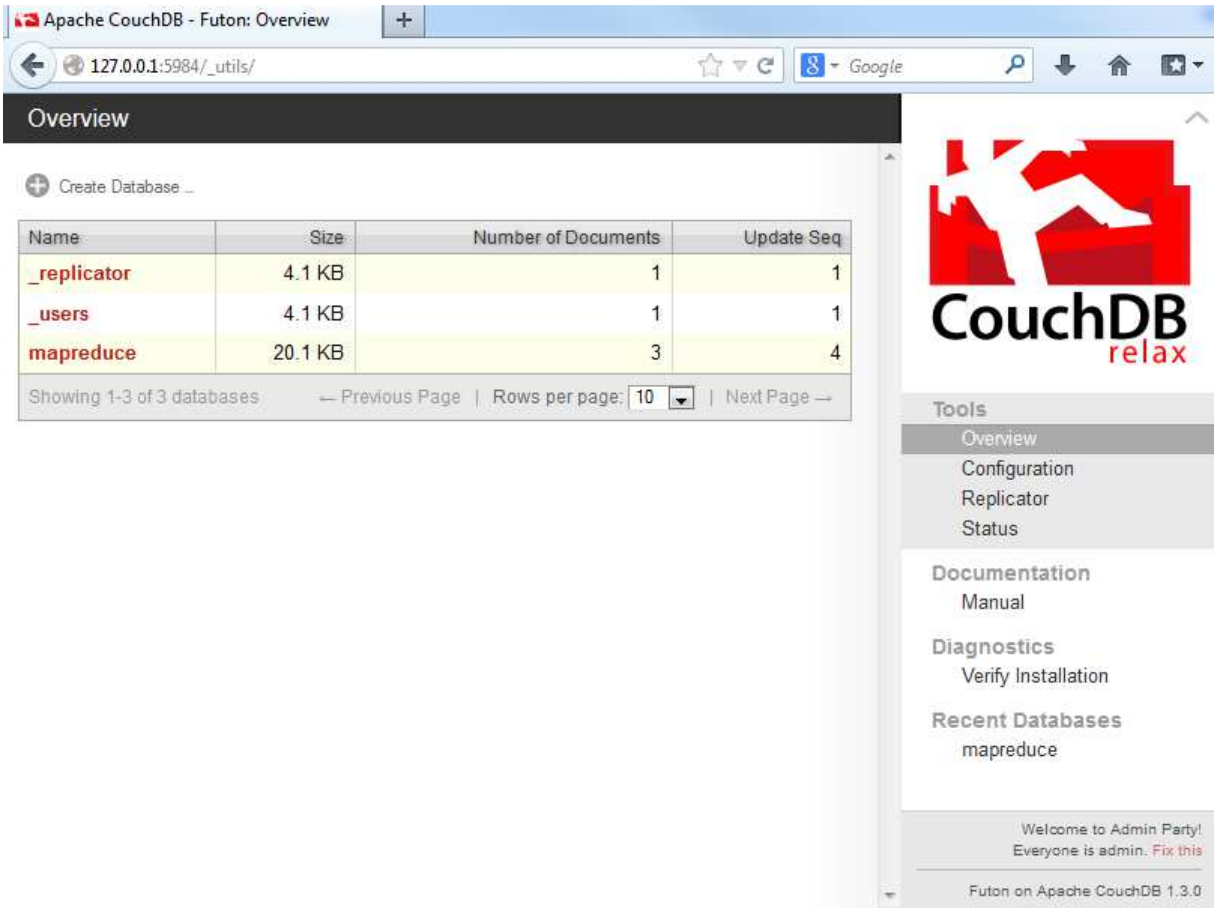

Abbildung 5 Futon

4.4 Design-Dokumente

Design-Dokumente sind ebenfalls Dokumente einer CouchDB-Datenbank. Im Vergleich zu anderen Dokumenten beginnt deren ID jedoch mit *_design/*. In erster Linie dienen sie dazu, den Quellcode der Anwendungen zu enthalten. Dieser Quellcode kann zum Beispiel MapReduce beinhalten, aber auch Listen- und Validierungsfunktionen. Eine weitere wichtige Funktion ist, dass man JavaScript- und HTML-Bibliotheken anhängen kann. Somit kann man größere Bibliotheken problemlos als Standalone halten. [Cou11]

4.4.1 MapReduce

MapReduce setzt sich aus Map und Reduce zusammen und ist ein Konzept, womit man eine Datenmenge aus der Datenbank holen und gegebenenfalls gruppieren kann. Im Vergleich zu SQL könnte man das Map als eine Art SELECT, und Reduce als GROUP BY betrachten. Map und Reduce sind zwei Phasen, welche nur nacheinander erfolgen können und in CouchDB

ausschließlich in JavaScript geschrieben. Die Map-Phase dient in erster Linie dazu, eine Datenmenge aus der Datenbank zu erhalten. Dabei werden alle Dokumente der Datenbank betrachtet. In der Reduce-Phase können die Ergebnisse der Map-Phase aggregiert werden [Cou11].

Das folgende Beispiel soll MapReduce besser veranschaulichen. Dafür wurden in einer Datenbank drei Dokumente angelegt:


```
{
  "_id": "id01",
 "_rev": "2-9dc63fafac163b9125b431f25ca88590",
  "patient": "Mustermann",
 "alter": 25
}

{
  "_id": "id02",
  "_rev": "1-762235e2246d23469b76865b32a40fd4",
  "patient": "Meyer",
  "alter": 43
}

{
  "_id": "id03",
  "_rev": "1-1231c021328bff5eb2fabb4572067d6a",
  "patient": "Meyer",
  "alter": 73
}
```


Wenn man nun eine gewisse Datenmenge aus der Datenbank erhalten möchte, muss man zunächst eine Map-Funktion erstellen. Dieser Funktion sollte immer das Dokument-Objekt als Parameter übergeben werden und mittels emit kann man bestimmen, welche Daten ausgelesen werden sollen. Emit arbeitet nach dem Key/Value Prinzip. In dem Beispiel soll der Key der Patient sein und als Value soll der Patient und sein Alter ausgegeben werden. Demnach könnte die Funktion wie folgt aussehen:

```
function(doc) {
  emit(doc.patient, {Patient: doc.patient, Alter: doc.alter});
}
```

Als Ergebnis erhalten wir dann folgende Abbildung:

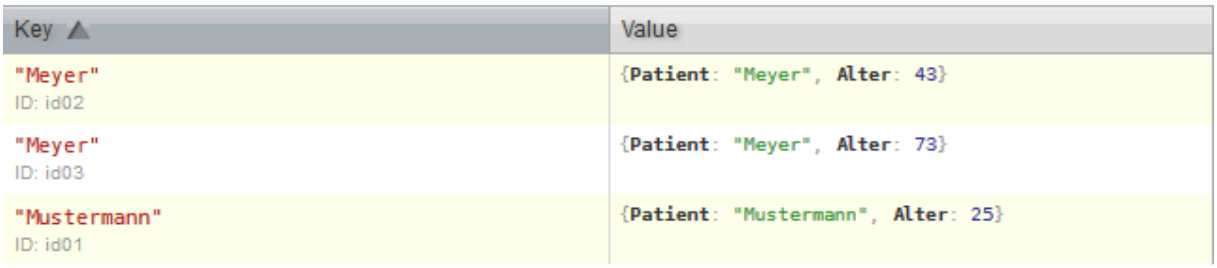

Abbildung 6 Ergebnis der Map-Funktion

Anschließend kann man sich mit Hilfe der Reduce-Funktion die ausgelesenen Daten gruppieren lassen. Dieser Funktion übergibt man die drei Parameter keys, values und rereduce und lässt sich als return die Summen der gefunden Values liefern. Die Funktion könnte dann beispielsweise so aussehen:

```
function(keys, values, rereduce) {
  if (rereduce) {
    return sum(values);
  } else {
    return values.length;
  }
}
```

Das daraus resultierende Ergebnis wäre dann folgende Abbildung:

Abbildung 7 Ergebnis der Reduce-Funktion

4.5 MVCC

Wenn man mit einer Datenbank arbeitet, möchte man die gespeicherten Daten immer in einem konsistenten Zustand vorliegen haben. Dies wird allerdings gefährdet wenn mehrere Zugriffe gleichzeitig auf die Datenbank erfolgen. In diesem Fall spricht man von konkurrierenden Zugriffen. Daher ist jedem Falle darauf zu achten, dass die Datenintegrität erhalten bleibt. Der klassischste Ansatz ist das Blocken von gleichzeitigen Schreibzugriffen. Das Prinzip ist recht simple. Solang ein Schreibezugriff stattfindet, werden alle anderen

Zugriffe geblockt (Abbildung 8). Diese Variante ist für die Datenintegrität sehr effektiv, jedoch birgt es einen gravierenden Nachteil im Bereich Performance.

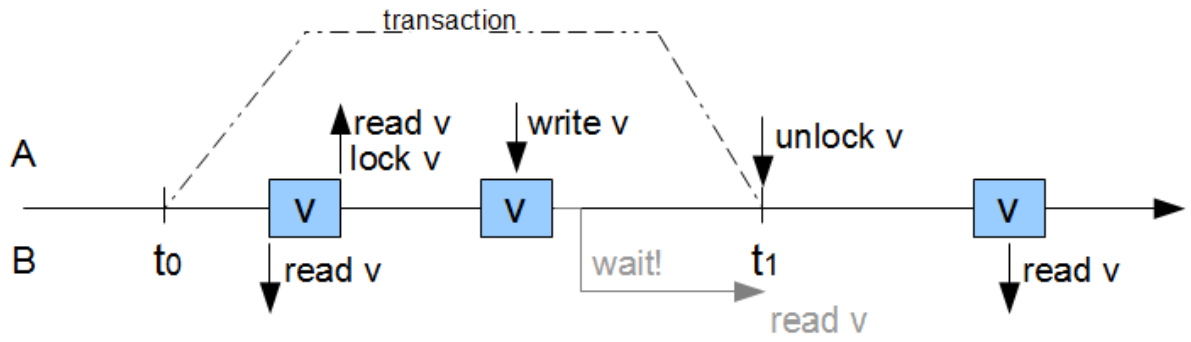

Abbildung 8 Klassisches Konkurrenzverfahren durch Sperren [Han11]

Für diesen Fall wurde das Multiversion-Concurrency-Control (MVCC)-Verfahren entworfen, welches in CouchDB integriert wurde. Dadurch soll genau das realisiert werden, was mittels dem klassischem Konkurrenzverfahren, nicht möglich war. Es können jeder Zeit Lesezugriffe erfolgen. Wie bereits erwähnt, besitzen Dokumente einer CouchDB-Datenbank eine Revisionsnummer, welche sich nach jeder Aktualisierung ändert. Durch MVCC sieht ein Benutzer, bei einer möglichen Aktualisierung seines Dokuments, einen aktuellen *Snapshot*. Während dieser Aktualisierung kann ein anderer Benutzer die Revision aufrufen und lesen (Abbildung 9). Es kommt erst zu einem Konflikt, wenn die Aktualisierung abgeschlossen werden soll und die letzte Revisionsnummer nicht mit der Revisionsnummer des aktuellen Dokuments übereinstimmt (Abbildung 10). Sollte dieser Konflikt eintreten, wird die gesamt Transaktion zurückgerollt und kann nochmals gestartet werden. [Cou11, Han11]

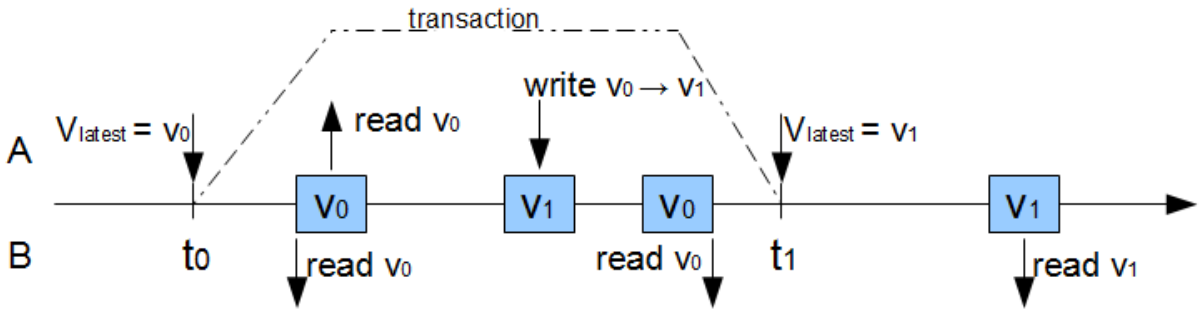

Abbildung 9 Sperren durch mehrere Versionen von v beim MVCC-Verfahren [Han11]

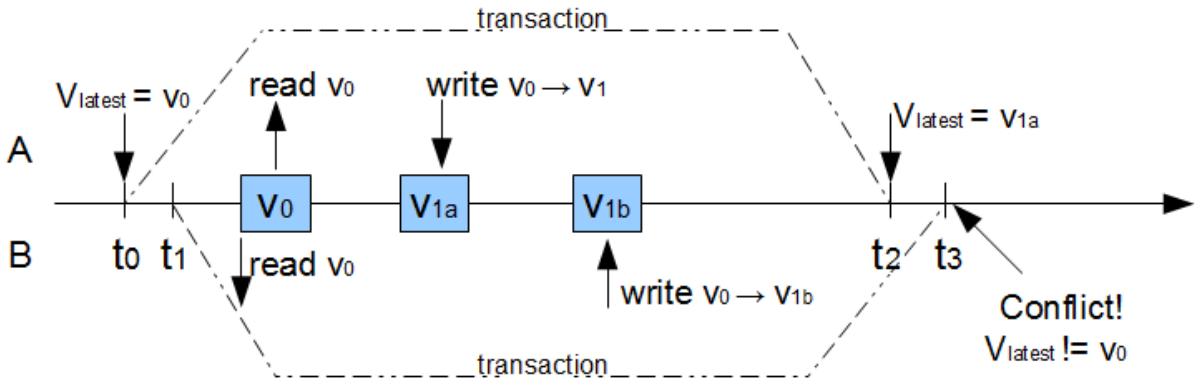

Abbildung 10 Konflikterkennung beim MVCC-Verfahren [Han11]

4.6 B+Tree-Index

In diesem Abschnitt geht es darum, wie die Daten strukturiert abgespeichert werden. Da dieser Teil sehr theoretisch ist, soll vorrangig nur das grobe Verständnis vermittelt werden. Der B+Tree-Index verwendet eine B-Baum Datenstruktur zur Organisation der Daten und strukturiert diese über einen Index. Der Index ist sozusagen der Identifier von den im B-Baum hinterlegten Daten. Ein B-Baum besteht aus einem Root-Knoten, welcher den Anfangsknoten beschreibt. Von ihm aus können mehrere Knoten abzweigen und wiederum Blattknoten besitzen. Die Beziehung zwischen Knoten und Blatt-Knoten wird auch häufig als Eltern-Kind-Beziehung definiert. Das bedeutet, dass die Kinder zu der Gruppe der Eltern gehören und ist quasi nur eine Aufgliederung des Elternknoten, um die Suche in einem B-Baum effizienter zu gestalten. Die folgende Abbildung (Abbildung 11) soll das Verständnis etwas einfacher machen. Die Suche in einem B-Baum erfolgt mit O(log n). In dem Beispiel in der Abbildung existieren 20 Datensätze. Daraus ergibt sich die Funktion log 20 zur Basis 2. Die Basis 2 kommt daher, weil in diesem Beispiel jeder Knoten maximal 2 Kinder besitzt. Wenn man diese Formel nun ausrechnet kommt auf einen Wert von circa 4,322. Das bedeutet, dass man 4-5 Suchschritte benötigt, um den gewünschten Index zu finden. Hätte jeder Knoten bspw. 4 Kinder, würde die Suche noch schneller erfolgen und man bräuchte nur noch 2-3 Suchschritte. [Cou11]

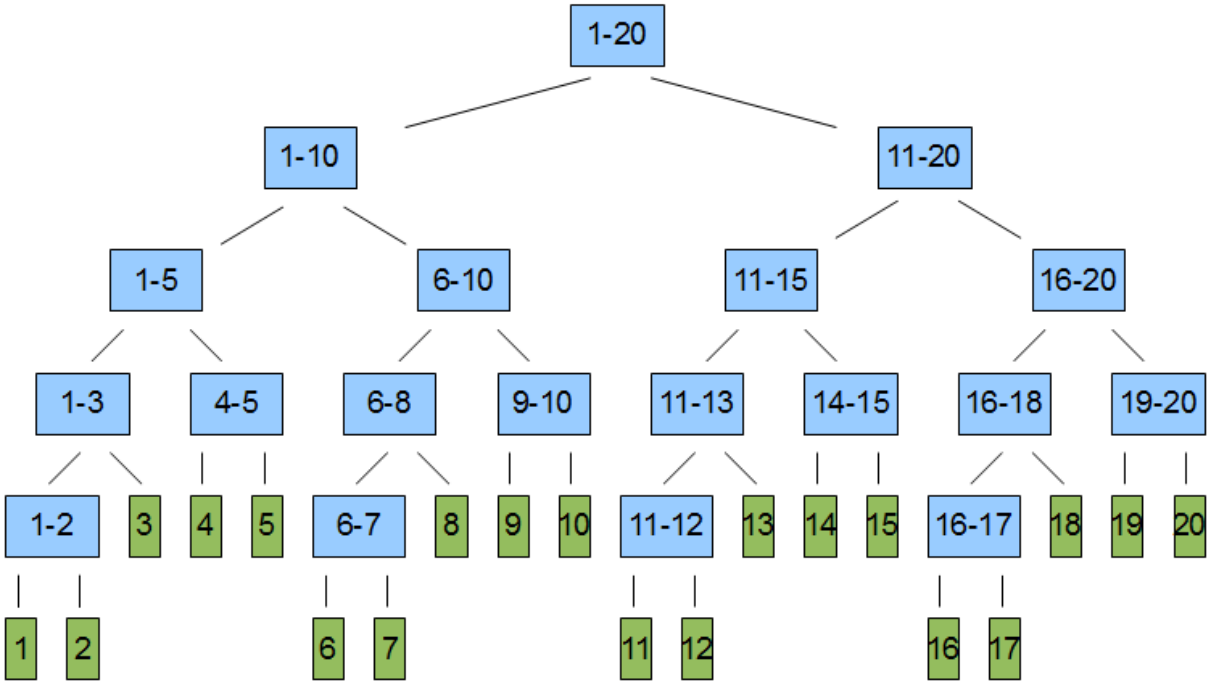

Abbildung 11 B-Baum mit 20 Datensätzen

So viel nun zum Thema B-Baum. In CouchDB wird eine Variante des B-Baum-Index verwendet, welche B+Tree-Index lautet oder auch B+Baum-Index. Diese Variante funktioniert ähnlich nur mit einer noch besseren Performance. Um dies zu erreichen verwendet man zwei verschiedene Indizes, speichert die Keys der Dokumente in den Knoten und die Daten in den Blattknoten. Die zwei verschiedenen Indizes lauten by_id_index und by_sequence_index. Wie bereits erwähnt besitzen die Dokumente in CouchDB eine eindeutige ID und eine Revisionsnummer. Diese beiden Keys sind relevant für die Indizes, denn der by_id_index benutzt die IDs als Knoten und der andere die Revisionsnummer. Da die Revisionsnummer bei jeder Aktualisierung eines Dokuments ändert, wird jedes Mal ein neuer Knoten im by_sequence_index erzeugt. Abschließend lässt sich sagen, dass der B+Tree-Index dadurch einen sehr effizienten Datenzugriff garantiert. [Cou11]

4.7 Das CAP-Theorem

Das CAP-Theorem kann ähnlich wie das ACID-Prinzip betrachtet werden. Jedoch nicht von der Funktionsweise, sondern eher von der Einsatzweise. CAP besteht genauso wie ACID aus verschieden Bausteinen, welche die Eigenschaften von Transaktionen in verteilten Systemen darstellen sollen. Die drei Bausteine aus denen CAP besteht, sind Consistency (Konsistenz), Availability (Verfügbarkeit) und Partition Tolerance (Ausfalltoleranz). Der Grundgedanke

besteht nun darin, dass in einem verteilten System immer zwei dieser drei Eigenschaften vorhanden sein müssen. Da man im Moment noch nicht alle drei gleichzeitig verwenden kann, muss man also immer auf eine Eigenschaft verzichten. Will man also Consistency und Partition Tolerance, so muss man auf Availability verzichten. Das folgende Szenario soll diese Aussage besser verdeutlichen. Dem Szenario liegt ein verteiltes System mit zwei Knoten zu Grunde. Der erste Knoten ist für Schreibzugriffe und der andere für Lesezugriffe zuständig. Beide greifen dabei auf dieselbe Datenbank zu. Wenn nun ein Schreibzugriff im Knoten1 erfolgt, wird die Datenbank aktualisiert und anschließend soll der Knoten2 per Nachricht informiert und aktualisiert werden. Hier kommen nun Partition Tolerance und Consistency ins Spiel. Durch Consistency wird garantiert, dass die Daten auch im Knoten2 konsistent sind bzw. aktualisiert werden. Und dadurch, dass Partition Tolerance vorhanden ist, kann selbst bei einem Ausfall einer Kommunikationsverbindung das System weiter arbeiten und auf Anfragen reagieren. In diesem Szenario wird allerdings auf Availability verzichtet, da in erster Linie auf die zwei anderen Eigenschaften geachtet wird und somit die Verfügbarkeit des Systems darunter leidet. Das soll jetzt aber nicht heißen, dass darauf komplett verzichtet wird. [Cou11]

4.7.1 CAP und CouchDB

Nachdem nun die Denkweise des CAP-Theorems bekannt ist, wird auch verständlicher, wie es in CouchDB umgesetzt wird. In CouchDB wird auf Partition Tolerance verzichtet, wodurch CouchDB kein verteiltes System ist. Es besteht zwar die Möglichkeit durch Funktionen, wie Replikation, eine Synchronisation zu anderen Knoten zu ermöglichen. Aber man spricht deswegen noch nicht gleich von einem verteilten System. Das bedeutet nun laut CAP, dass in CouchDB Consistency und Availability integriert sind. Im Falle von CouchDB steht das C allerdings für eventual consistent. Das kann bedeuten, dass ein Lesezugriff zunächst nicht das aktualisierte Ergebnis liefert. Jedoch liefert irgendwann jeder Lesezugriff das gleiche Ergebnis. Availability wird dadurch garantiert, dass beispielsweise ein Schreibzugriff auf den CouchDB-Server erfolgt und im Hintergrund automatisch auf einen freien Server repliziert wird. Dieses Verfahren ist auch unter dem Namen Master-Slave-Prinzip bekannt. [Cou11]

4.8 Resümee

Die wichtigsten Eigenschaften und Komponenten zu CouchDB sind nun bekannt und sollten das Grundverständnis der Arbeitsweise erläutert haben. In den nächsten Kapiteln werden die Unterschiede zwischen RDBMS und Document Stores, sowie die Umsetzung mit CouchDB an einem Beispiel erläutert.

5 Unterschiede RDBMS und Document Stores

Einige Unterschiede im Arbeiten mit Document Stores sind bereits im Kapitel 4 CouchDB sichtbar geworden. In diesem Kapitel werden die grundlegenden Unterschiede zwischen RDBMS und Document Stores erläutert.

5.1 Schemafreiheit

Ein wichtiger großer Unterschied zwischen Document Stores und RDBMS ist die Schemafreiheit. Was das bedeutet, soll in diesem Kapitel erläutert werden. Der wohl grundlegendste Unterschied besteht darin, dass bei RDBMS die Daten in Tabellen gespeichert werden, welche wiederum in Beziehung zu anderen Tabellen stehen können. Im Gegensatz dazu wurde bereits erklärt, dass die Daten Document Stores in Dokumenten gespeichert werden. Um dies besser zu verstehen, könnte man sagen, dass ein Datensatz einer Tabelle, einem Dokument entspricht. Besonders gut ist dies auch anhand der folgenden Abbildungen zu verstehen. In einer relationalen Datenbank wurden die drei Tabellen Patient, Arzt und Behandlung angelegt. Die Tabelle Behandlung soll eine Beziehung zwischen Patient und Arzt definieren, wodurch die Daten per Fremdschlüssel referenziert werden (Abbildung 13). Im Vergleich dazu erstellt man mittels Document Stores ein Dokument Behandlung, wo alle betreffenden Daten festgehalten werden können. Dies bringt natürlich den Vorteil, dass man Änderungen der Daten in einem Dokument vornimmt und nicht in jeder betreffenden Tabelle. Außerdem lassen sich Schemaänderungen leichter ausführen, als wie bei RDBMS. Bei RDBMS könnte man zum Beispiel über ALTER_TABLE eine Änderung vornehmen. Da

dieser Prozess allerdings meist zu lang dauert, sind RDBMS weniger geeignet für Webanwendungen. Dadurch dass Document Stores kein festes Schema besitzen, kann man unter anderem auch ganze Listen speichern (Abbildung 12). Jedoch muss man auch erwähnen, dass gerade im Beispiel Stammdaten, Relationen nützlich sind. Wenn ein Arzt, beispielsweise durch Heirat, einen neuen Namen annimmt, muss dieser nur in einer Tabelle geändert werden. Im Vergleich zu Document Stores müsste der Name in jedem Dokument geändert werden. Ein weiterer Unterschied ist, dass die Schemaänderungen bei Document Stores durch die Anwendung bestimmt werden können. In RDBMS muss dies immer im System selbst erfolgen. Inwieweit die eben genannten Unterschiede Vor- oder Nachteile sind, hängt allerdings immer von den Einsatzmöglichkeiten ab. Aber dazu später mehr unter Einsatzmöglichkeiten von NoSQL-Technologien.


```
{
   "_id": "beh_001",
   "_rev": "3-1034cf53cc2c4c088da3d09998576750",
   "PatientID": "pat_001",
   "PGebDatum": "29.01.1973",
   "ArztID": "arzt_001",
   "Arzt": "Dr. med. Viktor Meier",
   "ArztAnschrift": "Alaunstrasse 22, 01067 Dresden",
   "Patient": "Klaus Fischer",
   "PatientAnschrift": "Feldstrasse 52a, 01067 Dresden",
   "BehDatum": "30.06.2013",
   "BehArt": "Röntgen",
   "BehRaum": "Rö203"
}
```

```
{
   "_id": "beh_002",
   "_rev": "4-c5d7380233af77668c6c1bcaa09c4075",
   "PatientID": "pat_001",
   "PGebDatum": "29.01.1973",
   "PVorName": "Klaus",
   "ArztID": "arzt_001",
   "BehDatum": "30.06.2013",
   "BehArt": "Röntgen",
   "BehRaum": "Rö203",
   "PName": "Fischer",
   "PStrasse": "Aachener Strasse 52a",
   "PPLZ": "01067",
   "POrt": "Dresden",
   "ATitel": "Dr. med.",
   "AVorname": "Viktor",
   "AName": "Meier",
   "AStrasse": "Alaunstrasse 22",
   "APLZ": "01067",
   "AOrt": "Dresden"
}
```


Abbildung 12 JSON-Objekte in Dokumenten einer CouchDB

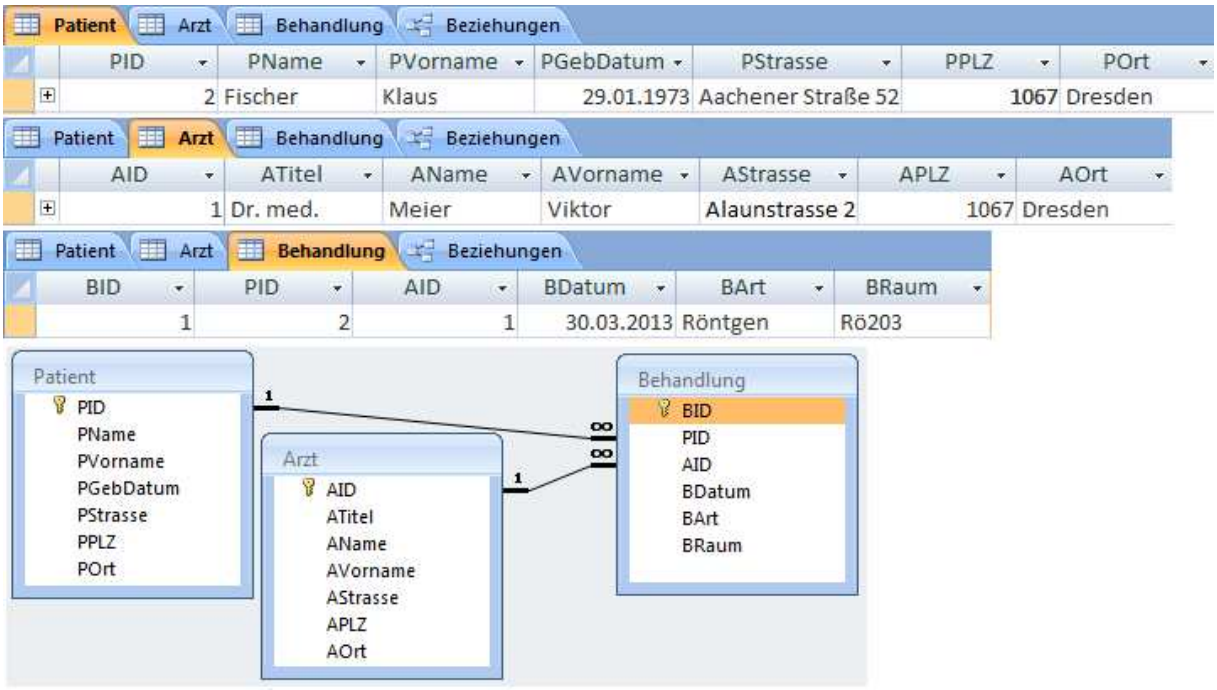

Abbildung 13 Tabellen und Beziehungen in RDBMS

5.2 Datenzugriff

Ein weiterer Unterschied besteht darin, wie die Systeme auf die Daten zugreifen. Sie unterscheiden sich zunächst aber schon darin, wie die eindeutigen Bezeichner, welche für den Datenzugriff durchaus relevant sind, definiert werden. Die Daten in einem RDBMS haben immer einen Primärschlüssel, welcher über die AUTO_INCREMENT-Funktion erzeugt werden können. Daraus resultiert allerdings, dass die Bezeichner nur in der Tabelle bzw. Datenbank eindeutig sind. Bei Document Stores hingegen bestehen die IDs meist aus einer Zeichenkette, die mit Hilfe eines Generierungsverfahrens erzeugt werden. Dadurch lassen sich zwei identische Schlüssel im gesamten System vermeiden.

Im Kapitel 4 CouchDB wurde bereits ein Datenzugriff für Document Stores mit MapReduce erläutert. Dies ist auch der weitverbreitetste Algorithmus für Document Stores. Für manche Vertreter gibt es aber auch eigens Abfragesprachen, um die Inhalte zu erhalten. Die meisten RDBMS nutzen die Datenbanksprache SQL. Um die Vor- und Nachteile besser zu erläutern, wurde für das jeweilige System ein Beispiel entworfen. In Quellcode 1 erfolgt der Datenzugriff auf eine relationale Datenbank mittels einer SQL-Abfrage. Die Anwendung wurde mit Java implementiert und über eine JDBC-Verbindung zur Datenbank realisiert. In Quellcode 2 wird ein Konsolenaufruf in Java gestartet, mit welchem die Daten über einen GET-Request aus

einer CouchDB-Datenbank geholt werden sollen. Beide Anwendungen arbeiten mit den Beispielen aus dem vorhergehenden Kapitel Schemafreiheit (Abbildung 12, Abbildung 13). Wenn man beide Anwendungen vergleicht, wird schnell deutlich, dass bei einer SQL-Abfrage eine viel aufwendigere Abfrage gestartet werden muss. Dies führt gerade in diesem Beispiel zu einem höheren Zeitaufwand, da eine Abfrage auf die Beziehungstabelle gestartet wird, und die Daten erst aus Patient und Arzt mittels JOIN referenziert werden müssen. Im Beispiel zwei hat man sofort alle Daten zu dem Dokument. Jedoch ist dies auch nur in diesem Beispiel so. Nimmt man das Beispiel aus 4.4.1 MapReduce, sieht man, dass für Document Stores durchaus komplizierterer Abfragen schwieriger zu realisieren sind. Zusammenfassend lässt sich also sagen, dass der Datenzugriff in einem RDBMS über eine SELECT-Abfrage geschieht, und bei Document Stores über HTTP-Methoden, MapReduce oder andere Abfragesprachen erfolgt.

```
public static void main(String[] args) {

    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=D:/Datenbank1.mdb","","");
        stmt = con.createStatement();
        String arg0 = "Select * From (Behandlung As b INNER JOIN Patient As p ON b.PID = p.PID) INNER JOIN Arzt As a ON b.AID = a.AID";
        rs = stmt.executeQuery(arg0);
        while(rs.next()){
            //do something
        }
    }
    catch(Exception e){
    e.printStackTrace();
    }
}
```

Quellcode 1 Datenbankzugriff in Java auf Accessdatenbank

```
//Command erzeugen
setFile("c:/users/master/desktop/bachelor/curl/");
//HTTP-Methode zum Holen der Daten
setGet("http://127.0.0.1:5984/behandlung/beh_002");
String command = "cmd /c start cmd.exe /K \"cd " + getFile() + " && curl.exe " + getGet() + ">ausgabe.txt";
try {
    setP(Runtime.getRuntime().exec(command));
    Thread.sleep(3000);
    getP().destroy();
}
catch(Exception e) {}
```

Quellcode 2 Datenbankzugriff auf CouchDB über Konsolenaufruf GET-Request

5.3 Horizontale Skalierbarkeit

In Datenbanken spielt die Skalierbarkeit eine große Rolle. Bei der Skalierbarkeit geht es darum, die Leistung eines Systems zu steigern. Vertikale Skalierbarkeit beschreibt das Hinzufügen von Ressourcen, welche Speicherplatz, CPU oder auch Grafik steigern können. Jedoch stößt diese häufig an ihre Grenzen, da die besten Ressourcen immer teurer werden

und ein System irgendwann ein Limit erreicht. Durch horizontale Skalierbarkeit kann man einem System neue Rechner oder Knoten hinzufügen. Dies bedeutet, dass man günstigere Server verwenden und miteinander verbinden kann. Allerdings ist dies von dem System abhängig, ob es ein verteiltes System mit mehreren Knoten zulässt.

Ein RDBMS ist eine 1-Server-Hardware, wodurch nur ein Server zulässig ist. Das heißt, dass ein RDBMS nicht horizontal skalierbar ist. Die Begründung liegt ganz einfach darin, dass durch Primär- und Fremdschlüssel aufwendige JOINS benötigt werden und somit lassen sich die Daten nur schwer und sehr aufwendig auf mehrere Knoten verteilen. Da ein Document Store seine Daten in Dokumente speichert, lassen sich diese natürlich einfacher auf mehrere Knoten verteilen. Im Falle von CouchDB wurde bereits gesagt, dass es dort noch nicht möglich ist, prinzipiell wäre es aber möglich. Schlussendlich sind beide Systeme zwar vertikal skalierbar, jedoch verfügen nur die Document Stores über horizontale Skalierbarkeit, was natürlich einen erheblichen Vorteil mit sich bringt.

6 Einsatzmöglichkeiten von NoSQL-Technologien

In diesem Kapitel sollen nun mit Hilfe der bisherigen Kenntnisse mögliche Einsatzgebiete für NoSQL-Technologien, speziell mit CouchDB, ermittelt werden. Dafür wurden zunächst Einsatzgebiete im Umfeld von KIS ermittelt. Anschließend wurde an einem sehr komplexen Szenario mit Hilfe von Testimplementierungen analysiert, inwiefern die NoSQL-Technologien wirklich Sinn machen.

6.1 Mögliche Einsatzgebiete

Zunächst braucht man natürlich als Grundlage der folgenden Analysen die Kenntnisse darüber, welche Einsatzmöglichkeiten in einem Krankeninformationssystem, sowie in dessen Umfeld existieren. Wie bereits gesagt, besteht ein KIS aus mehreren Anwendungs-bausteinen. In der folgenden Tabelle (Tabelle 2) werden die Bausteine mit ihrer Funktion benannt. Dabei wäre zu erwähnen, dass nicht alle Systeme vorkommen werden, die auch existieren. Jedoch werden die wichtigsten und die am häufigsten eingesetzten Systeme benannt.

Bausteine	Funktion
PVS	Das Patientenverwaltungssystem wird zur Aufnahme, Verlegung und Entlassung eines Patienten eingesetzt und enthält die Patienten-stammdaten, welche über eine eindeutige PIN definiert werden. Außerdem enthält es zu jedem Patient die Falldaten. [Anw13]
KDMS	Das Klinische Dokumentations- und Managementsystem wird auf den Stationen zur klinischen Dokumentation eingesetzt. Mit dessen Hilfe können die Inhalte der Patientenakte verwaltet werden. [Anw13]
RIS	Das Radiologieinformationssystem wird in der Radiologie eingesetzt. Es dient der Planung und Verwaltung der Untersuchungstermine, sowie der Organisation des Untersuchungsablaufes. Die dafür benötigten Patientendaten können bereit gestellt werden. Außerdem dient es der Erstellung der Befunde. [Anw13]
PACS	Das Bildspeicher- und Kommunikationssystem ist das Archivsystem des RIS und dient der langfristigen Aufbewahrung digitaler Bilder und Befunde. [Anw13]
LIS	Das Laborinformationssystem dient dem Labor zur Ablaufsteuerung bei der Analyse einer Probe, bei der die Ergebnisse dokumentiert und anschließend Befunde erstellt werden. Alle patientenbezogenen Daten können hier ebenfalls wieder bereit gestellt werden. [Anw13]
PDMS	Das Patientendatenmanagementsystem wird auf der Intensivstation eingesetzt. Es wird zur Überwachung eines Patienten benötigt. So können bspw. Vitalwerte automatisch aufgezeichnet, ausgewertet und übersichtlich dargestellt werden. [Anw13]
KS	Ein Kommunikationsserver stellt die Kommunikation zweier Anwen-dungsbausteine eines KIS her.

Tabelle 2 Anwendungsbausteine eines KIS

Zur Veranschaulichung eines KIS soll folgende Abbildung dienen. Sie zeigt wie die einzelnen Subsysteme in Verbindung stehen, beziehungsweise wie sie miteinander kommunizieren.

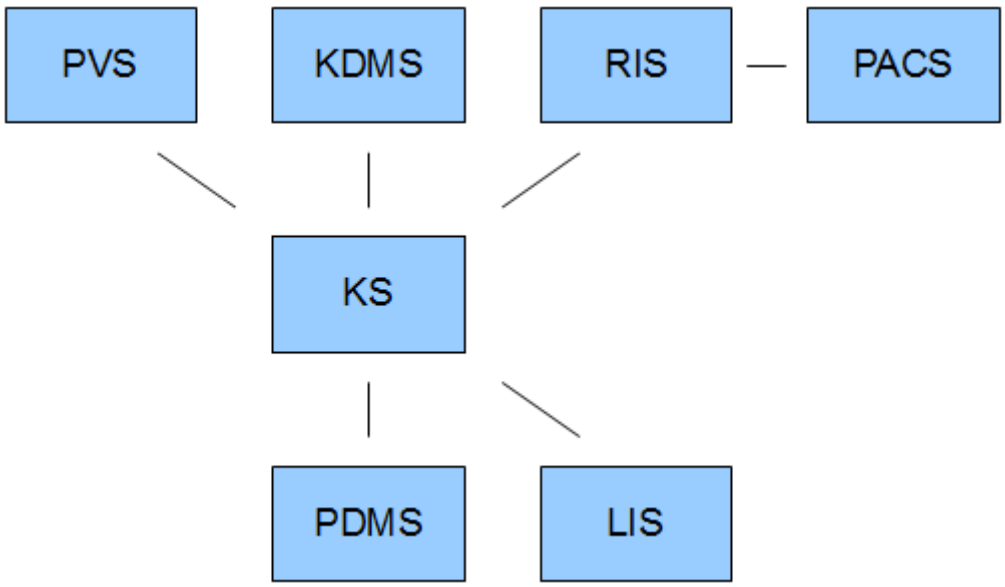

Abbildung 14 Aufbau eines KIS

Da nicht nur die Anwendungsbausteine eines KIS eine Rolle spielen, sondern auch die Einsatzgebiete in dessen Umfeld, wurden verschiedene Möglichkeiten recherchiert und aufgelistet. Das Umfeld beschreibt die informationsverarbeitenden Prozesse der medizinischen Versorgung außerhalb eines KIS.

Umfeld	Beschreibung
Apotheke Pflege	In einer Apotheke werden Arzneimittel und Medizinprodukte vertrieben und zum Teil auch hergestellt.
Niedergelassener Arzt	Er arbeitet meist in seiner eigenen Arztpraxis, in welcher er seine Patienten behandelt.
Radiologie	Ist eine Einrichtung, in der radiologische Diagnostik ausgeübt wird.
Labor	In einem Labor, oder auch Laboratorium, können praktisch Experimente durchgeführt werden, welche zur Diagnostik benötigt werden.
Ambulante Pflege	In der Ambulanten Pflege werden Patienten häuslich gepflegt und versorgt.
Rettungsdienst	Der Rettungsdienst ist rund um die Uhr für medizinische Notfälle abrufbar.
Krankentransport	Der Krankentransport dient zur Krankenbeförderung.
Spendedatenbanken	Hiermit sind Datenbanken gemeint, die Spenden, wie zum Beispiel die Deutsche Knochenmarksspenderdatei, enthalten.

Physiotherapie	Therapiert die Bewegungs- und Funktionsfähigkeit des Körpers durch äußerliche Heilpraktiken.
Robert-Koch-Institut	Bundesinstitut für Infektionskrankheiten
Gesundheitsbehörden	Institute, die die medizinische Versorgung kontrollieren.
Kassenärztliche Vereinigung	Sie umfasst alle Ärzte und Therapeuten, die sich der ambulanten Behandlung von gesetzlich Krankenversicherten zugewendet haben und dazu ermächtigt sind.

Tabelle 3 Umfeld von KIS

6.2 Szenario

In diesem Szenario wird ein Komplexbeispiel erläutert, wie der Ablauf in einem KIS aussehen könnte. Das KIS ist so aufgebaut wie in der Abbildung 14. Nachdem die einzelnen Bausteine bereits vorgestellt wurden, sollten die folgenden Ausführungen verständlich sein.

Ein Patient hat sich einen Bänderriss im Sprunggelenk zugezogen, wodurch er stationär im Krankenhaus operiert werden soll. Der Patient ist zum ersten Mal in diesem Krankenhaus, wodurch er zunächst neu angelegt werden muss. Anschließend wird er auf die Chirurgie verlegt, wo vor der Operation noch einmal das betroffene Sprunggelenk geröntgt und ein Bluttest veranlasst wird. Nach der Operation liegt der Patient kurzzeitig auf der Intensivstation zur Beobachtung. Wenn die Vitalwerte wieder konstant in Ordnung sind, soll er wieder auf die Chirurgie gelegt werden. Nach einer Woche wird der Patient entlassen.

Der genaue Ablauf wird in der folgenden Abbildung genauer beschrieben:

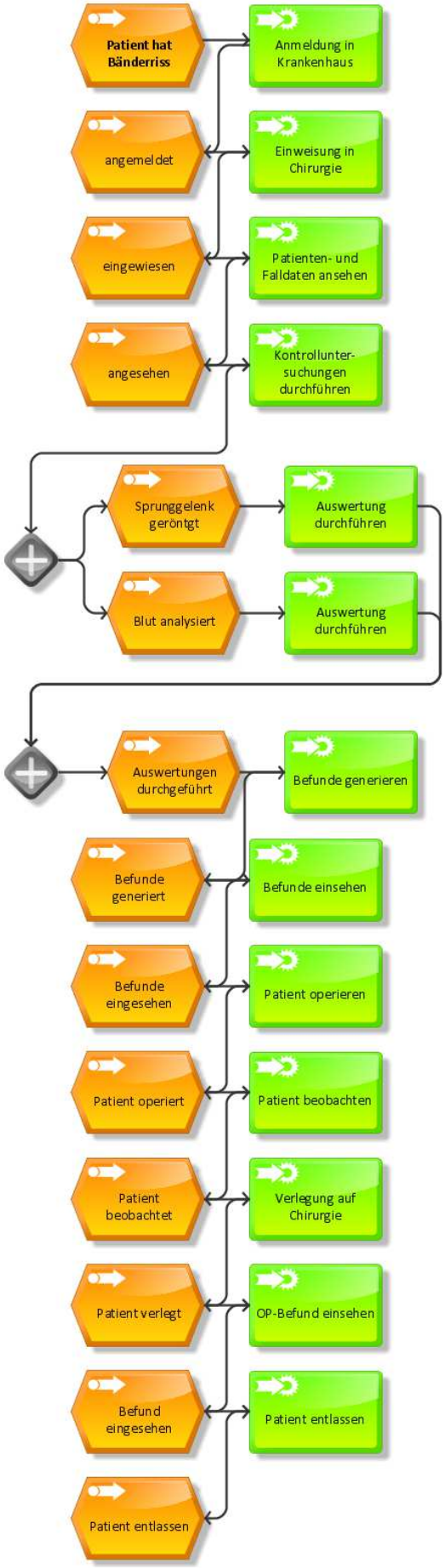

Abbildung 15 Szenario 1 - Legende im Anhang (Abbildung 27)

6.2.1 IST-Zustand mit RDBMS

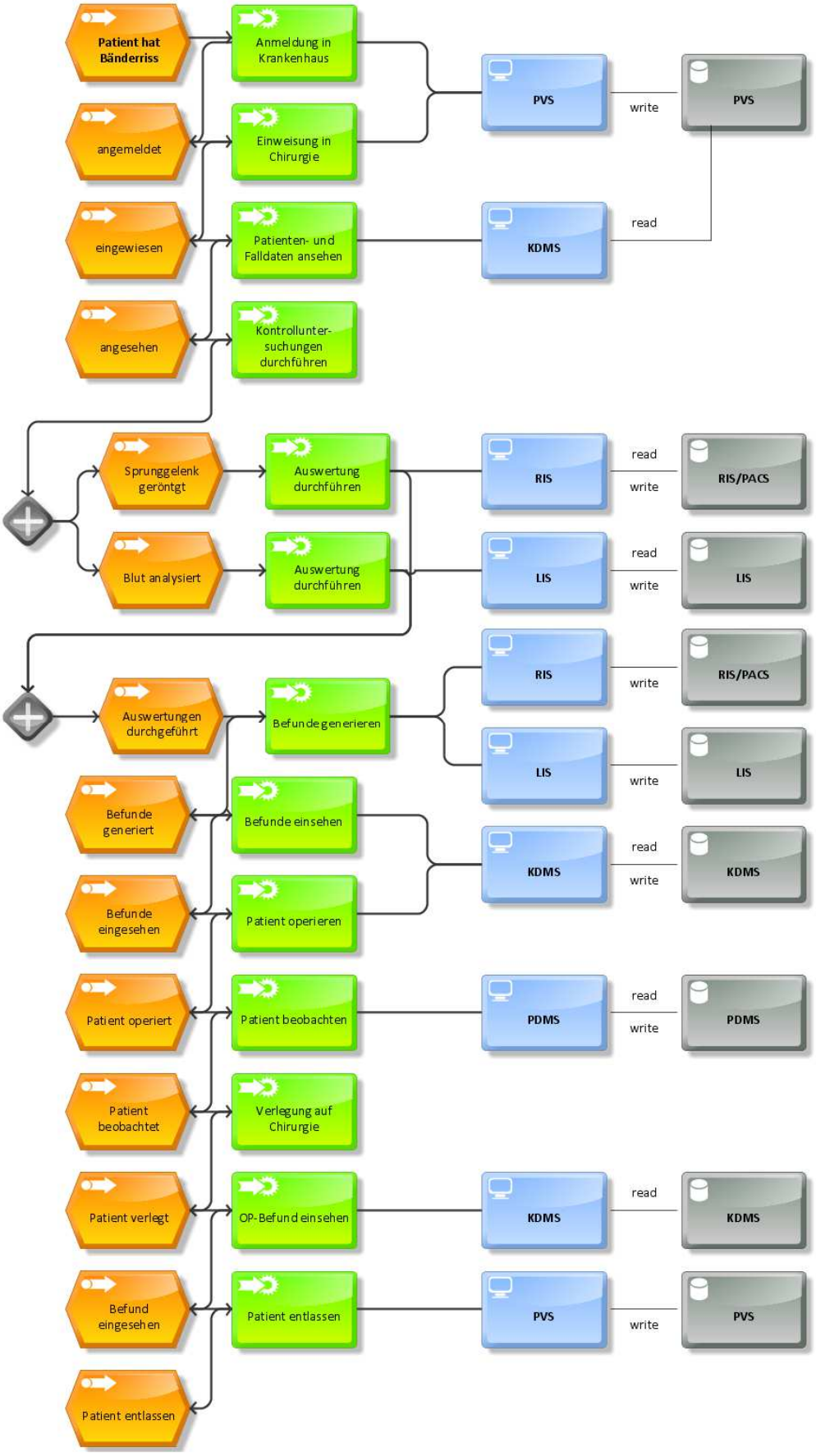

Abbildung 16 IST-Zustand mit RDBMS

6.2.2 Realisierung mit CouchDB

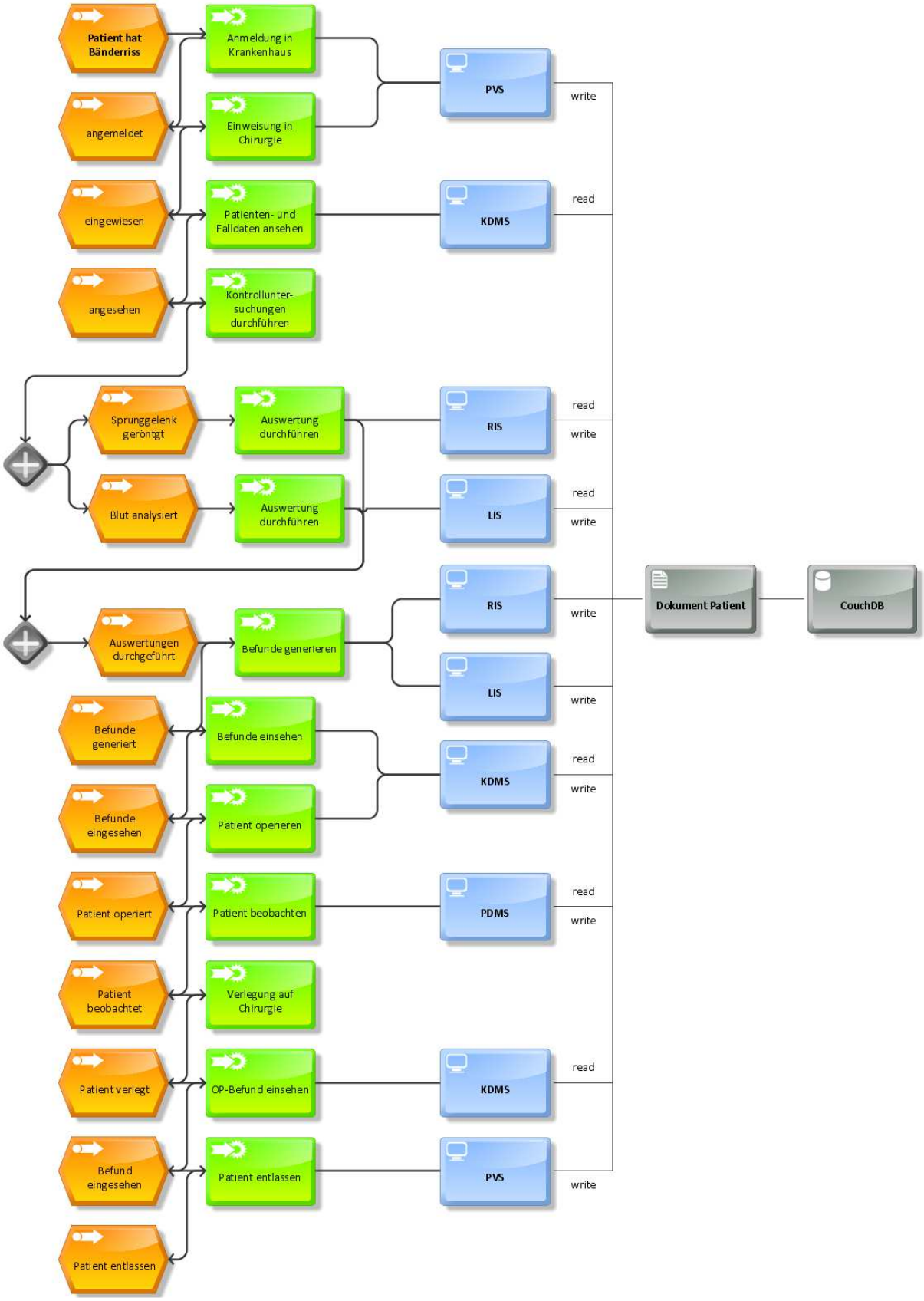

Abbildung 17 Implementierung mit CouchDB

6.2.3 Vergleich von 6.2.1 und 6.2.2

Der IST-Zustand mit RDBMS von Punkt 6.2.1 beschreibt die Datenverarbeitung mit Hilfe der Anwendungsbausteine eines KIS. Dabei fällt auf, dass jedes Anwendungssystem seine eigene Datenbank hat um Datenredundanz zu vermeiden. Dafür müssen die Anwendungssysteme miteinander kommunizieren können, um auch die Daten der anderen Systeme lesen zu können. Nimmt man zum Beispiel Aktivität 3 aus dem Szenario: Nachdem der Patient eingewiesen wurde, sollen auf der Station die Patientenstammdaten und Falldaten im KDMS eingesehen werden. Die Daten wurden allerdings bei der Anmeldung, in der Datenbank des PVS angelegt. Daher muss an dieser Stelle eine aufwendigere Funktion erstellt werden, da die Daten mit den Patientendaten der KDMS-Datenbank verknüpft werden müssen. Dasselbe geschieht überall da, wo ein System mit einem anderen kommunizieren möchte. Es wird eben solch eine komplexere Suchfunktion mittels SQL benötigt, um die Daten überall da, wo sie benötigt werden auch zu erhalten.

Die mögliche Realisierung mit CouchDB von Punkt 6.2.2 könnte so aussehen, dass beispielsweise jeder Patient sein eigenes Dokument in der Datenbank von CouchDB erhält. Wie in dem Szenario zu erkennen, würde dann nur eine zentrale Datenbank benötigt werden, in welcher die Dokumente liegen. Wenn man diesmal Aktivität 5-6 betrachtet, würden das RIS und LIS in dasselbe Dokument schreiben, und nicht nur lesen.

Der grundlegende Unterschied zwischen den beiden Szenarien wäre also, dass nicht mehr jedes System seine eigene Datenbank benötigt. Was hat das aber für Konsequenzen? Durch RDBMS hat man sehr viele Relationen in einem KIS, wodurch immer wieder Datenbanken mit einer anderen verknüpft werden müssen und somit ein hoher Leistungsaufwand entstehen kann. Mit Hilfe von CouchDB würde in diesem Szenario jeder Patient sein eigenes Dokument besitzen, quasi wie eine Art Patientenakte. Dadurch jeder Patient eine eindeutige PIN hat, kann man jedes Dokument auch genau einem Patient einwandfrei zuordnen. Was aber im Vergleich zu RDBMS bleibt, wären die unterschiedlichen Anwendungssysteme. Dadurch jedes System anders arbeitet bzw. arbeiten muss, ist es an dieser Stelle sinnvoller, unterschiedliche Bausteine zu implementieren, die aber jeweils über die PIN des Patienten auf das jeweilige Dokument zugreifen können. Ein paar dieser Bausteine wurden zu diesem Zweck mit den wesentlichsten Funktionen implementiert.

Die unterschiedlichen Konkurrenzverfahren machen sich auch in den Szenarien bemerkbar. Bei RDBMS kann es passieren, dass durch das klassische Konkurrenzverfahren auf ganze Datensätze oder Tabellen nicht zugegriffen werden können. Die Dokumente einer CouchDB sind jederzeit lesbar.

Die weiteren Unterschiede sind die, welche schon in Kapitel 5 beschrieben wurden. Daher sollte durch die Implementierung getestet werden, ob die drei wesentlichen Unterschiede (Schemafreiheit, Datenzugriff und Horizontale Skalierbarkeit) auch in der Praxis zutreffen. Wobei, wie bereits schon geschildert die horizontale Skalierbarkeit in CouchDB noch nicht direkt umgesetzt werden kann. Im folgenden Kapitel wird die Implementierung der Anwendungen erläutert.

6.2.5 Implementierung der Anwendungen

Der Datenzugriff wurde so realisiert, dass man in Java einen Konsolenaufruf startet, wodurch die HTTP-Methoden benutzt werden können. Beim Schreiben von Daten, kann durch die Variabilität von JSON-Objekten die Schemafreiheit geschaffen werden. Das bedeutet, dass man dem Dokument hinzufügen möchte, was man will. In RDMBS hat man immer ein festes Schema bzw. wird das Schema durch die Datenbank festgelegt. In den Implementierungen wird das Schema durch die Anwendung definiert. Dadurch nun jeder Patient sein eigenes Dokument hat, kann auch die Horizontale Skalierbarkeit gewährleistet werden. Da alle Daten zu dem Patienten in diesem Dokument stehen, lassen sich die Dokumente auf mehrere Knoten aufteilen, ohne dass dies Auswirkungen auf die einzelnen Dokumente hat, da sie nicht in Beziehung zueinander stehen.

Es wurden Prototypen für ein PVS, KDMS und RIS implementiert. Wie bereits gesagt, sollten sie die Einsatzmöglichkeiten und Grenzen von NoSQL-Technologien im Umfeld von KIS aufzeigen. Die Prototypen werden nun im Einzelnen mit ihren Funktionen vorgestellt. Der Quellcode zu den Prototypen ist im Anhang beziehungsweise auf der beiliegenden CD zu finden.

PVS

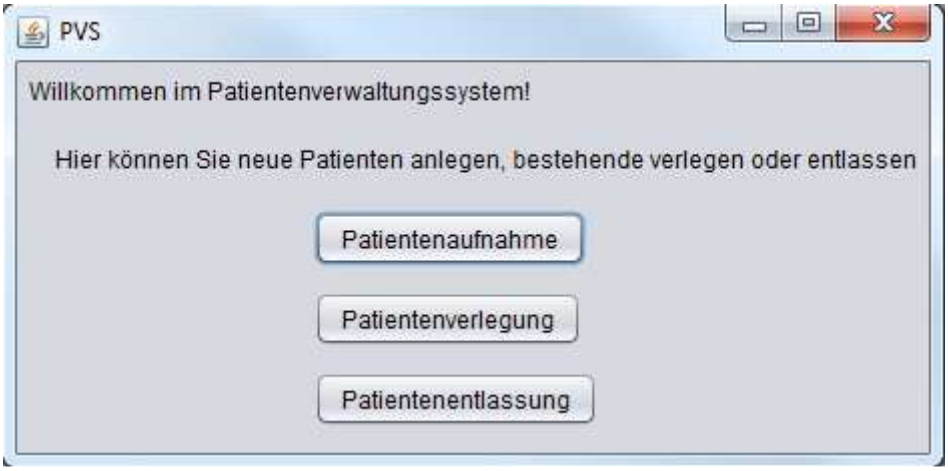

Abbildung 18 Prototyp PVS

In diesem Prototypen des PVS wurden die drei wesentlichen Funktionen Patienten-aufnahme, -verlegung und -entlassung implementiert. Aus dem Startmenü in Abbildung 18 gelangt man über Buttons in die jeweiligen Anwendungen. In Abbildung 21 sieht man die Patientenaufnahme des Prototypen. Die Klasse wurde so konzipiert, dass die Labels und Textfelder, die Key/Value Paare im Dokument bilden. Die Daten werden dabei in ein JSON-Objekt gepackt und anschließend mit einem PUT-Request über die Eingabeaufforderung an die Datenbank geschickt. Jedoch darf der Patient an dieser Stelle noch nicht in der Datenbank vorhanden sein. Die Patientenverlegung in Abbildung 22 arbeitet ähnlich. Zuvor muss der Patient jedoch bereits in der Datenbank vorhanden sein. Dafür wird mit einem GET-Request die Datenbank nach der eingegebenen ID überprüft. Wenn er vorhanden ist, werden außerdem die Daten des Patienten in einem Textdokument zwischengespeichert. Anschließend werden die geholten Daten, mit den neu eingegebenen Daten zu einem JSON-Objekt verknüpft und mit einem PUT-Request wiederum an die Datenbank gesendet. Dabei ist zu beachten, dass immer die aktuelle Revisionsnummer aus den geholten Daten gefiltert wird, da diese für den PUT-Request eines bereits vorhandenen Dokuments benötigt wird. Die neu eingegebenen Daten werden dann anschließend einfach dem Dokument als Key/Value Paar angehangen. Die Patientenentlassung funktioniert nach demselben Prinzip wie die Verlegung (Abbildung 23).

KDMS

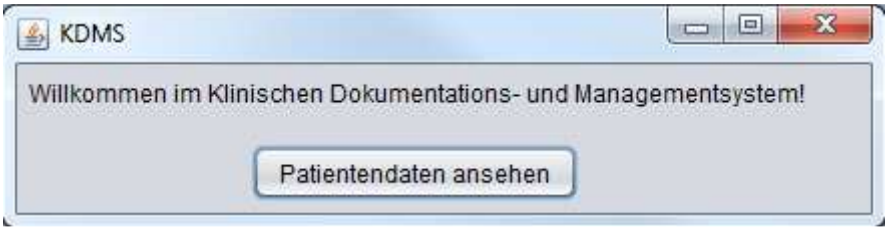

Abbildung 19 Prototyp KDMS

Im KDMS sollen die Daten eines Patienten eingesehen werden können. In Abbildung 24Abbildung 19 kann man dafür zunächst wieder nach dem Patienten suchen. Das davon gelieferte Objekt ist wieder ein JSON-Objekt, woraus die Key/Value-Paare ausgelesen und in einer Tabelle übersichtlich dargestellt werden (Abbildung 25). Die Suche nach dem Patienten erfolgt wieder nach demselben Prinzip wie zuvor im PVS.

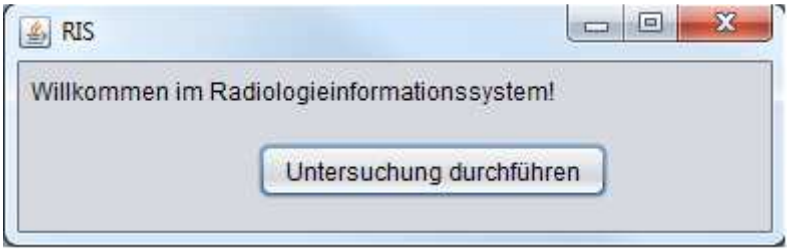

Abbildung 20 Prototyp RIS

Im dritten Prototypen, dem RIS, kann für einen in der Datenbank vorhandenen Patienten eine Untersuchung angelegt werden. Desweiteren gibt es die Möglichkeit eine Datei, beispielsweise ein Radiologiebild, hochzuladen (Abbildung 26). Dies geschieht ebenfalls über ein PUT-Request und wird dem Dokument des Patienten hinzugefügt.

6.2.6 Resümee

Prinzipiell lässt sich sagen, dass hier ein Ansatz geschildert wird, wie mit Hilfe von CouchDB, NoSQL-Technologien im KIS eingesetzt werden könnten. Da dieses Beispiel sehr komplex ist und versucht möglichst viele Bereiche beziehungsweise Szenarien im KIS abzudecken, werden Einsatzmöglichkeiten und auch Vor- sowie Nachteile von CouchDB bekannt. Die Anwendungen sind nützlich und zeigen, dass grundlegend ein Einsatz dieser Technologien stattfinden kann. Jedoch stellt sich die Frage, inwieweit die Schemafreiheit in einem KIS sinnvoll ist. Durch die Schemafreiheit der CouchDB kann in jedem Anwendungssystem auf dasselbe Dokument geschrieben werden. Dies hat zur Folge, dass auf den unterschiedlichen

Systemen Verwirrung herrschen kann, da man nicht mehr einwandfrei nachvollziehen kann, was ein anderer Benutzer geändert hat. Dies hat außerdem zur Folge, dass die Implementierung erheblich erschwert wird, wenn die Schemafreiheit in der Anwendung liegt. So kann man zum Beispiel Listeneinträge schwieriger auslesen. Abschließend lässt sich sagen, dass es einen möglichen Ansatz gibt, RDBMS in einem KIS zu ersetzen, aber mit den momentanen Funktionen noch nicht einwandfrei realisierbar für solch eine große Menge an Daten und Relationen ist.

7 Zusammenfassung

Das Ziel dieser Arbeit war es, mögliche Einsatzgebiete von NoSQL-Technologien im Umfeld von KIS herauszufinden. Dafür wurden zunächst die Unterschiede zwischen einem RDBMS und Document Stores aufgezeigt und anschließend mit dem typischen Vertreter, namens CouchDB, geeignete Implementierungen durchgeführt. Die Unterschiede zeigten, was die Document Stores versuchen besser zu machen wie die RDBMS. Durch die Implementierung der Prototypen wurden die Unterschiede nicht nur getestet, sondern auch neue Möglichkeiten aufgezeigt, dies umzusetzen. Zu Beginn der Arbeit wurde die Implementierung noch mit Hilfe von HTML und JavaScript getestet. Jedoch entwickelte man im Laufe der Arbeit die Anwendungen mit Hilfe von Java, um die Anwendungen dynamischer zu gestalten. Außerdem können eigene Algorithmen zur Datenverschlüsselung mehr Freiheiten in Bezug auf die Flexibilität der Anwendung schaffen. Weitestgehend lässt sich sagen, dass der Einsatz von NoSQL-Technologien im Umfeld von KIS möglich ist. Jedoch werden sie vorerst strittig bleiben, da sie in ihrer Entwicklung noch im Entwicklungsstadium sind. Allerdings besitzen sie bereits sehr gute Eigenschaften, um den RDBMS ernsthafte Konkurrenz zu machen.

Eine Schwäche, die bei der Implementierung auffiel ist, dass bei einer Aktualisierung des Dokuments alle bereits vorhandenen Daten im Dokument überschrieben werden könnten. Das heißt, dass die vorhandenen Daten vorher ausgelesen, und anschließend mit den neuen Daten als ein JSON-Objekt wieder an das Dokument gesendet werden müssen. Das hat zur Folge, dass die zu sendenden im größer werden, je mehr Daten sie enthalten und einen höheren Aufwand für die Implementierung bedeuten.

Die vorgestellten Prototypen decken nur grundlegende Funktionen ab. So ist es momentan zum Beispiel noch nicht möglich Umlaute in den JSON-Objekten zu benutzen. Für die Zukunft wäre es interessant die Anforderungen an ein Anwendungssystem genauesten zu studieren und dementsprechend einen Prototyp weiter auszubauen. So könnte man sich noch besser ein Bild davon machen, ob ein NoSQL-System ein RDBMS vollständig ersetzen könnte.

I Literaturverzeichnis

[Anw13] Beschreibung der Anwendungsbausteine eines KIS – Ausschnitt der Diplomarbeit von Sabine Lehmann – Uni Leipzig *3LGM²-basiertes Referenzmodell für die digitale Archivierung von Patientenunterlagen.* JSP, 2013. www.3lgm2.de/Baukasten/Baukasten-Unterstützung/Use_Cases/Digitale_Archivierung/Beschreibung-Anwendungsbausteine.jsp zuletzt besucht am 05.08.2013

[Cou11] Andreas Wenk, Till Klampäckel: *CouchDB – Das Praxisbuch für Entwickler und Administratoren.* Galileo Press Bonn, 2011

[Cou13] Sergej Sintschilin – Seminar an Uni Leipzig: *Document Stores – CouchDB.* PDF, 2013. http://dbs.uni-leipzig.de/file/seminar_1112_sintschilin_ausarbeitung.pdf zuletzt besucht am 02.07.2013

[FHK13] FH Köln – Datenbanken Online Lexikon: *NoSQL Grundlagen*. Webseite, 2013. http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/NoSQL zuletzt besucht am 02.07.2013

[Han11] *NoSQL – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* Carl Hanser Verlag München, 2011

[Wik13a] WIKIPEDIA – Die freie Enzyklopädie: *RDBMS.* Webseite, 2013. http://de.wikipedia.org/wiki/RDBMS zuletzt besucht am 05.08.2013

[Wik13b] WIKIPEDIA – Die freie Enzyklopädie: *Transaktion – ACID-Prinzip.* Webseite, 2013. http://de.wikipedia.org/wiki/Transaktion_(Informatik)#ACID-Prinzip zuletzt besucht am 05.08.2013

[Wik13c] WIKIPEDIA – Die freie Enzyklopädie: *Krankenhausinformationssystem.* Webseite, 2013. http://de.wikipedia.org/wiki/Krankenhausinformationssystem zuletzt besucht am 05.08.2013

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Quellcodeverzeichnis

V Glossar

Abkürzungsverzeichnis

API	-	Application Programming Interface
BLOBS	-	Binary Large Object
CD	-	Compact Disc
CouchDB	-	Cluster of unreliable commodity hardware Data Base
DBM	-	Database Manager
HTTP	-	Hypertext Transfer Protocol
ID	-	Identifikator
JSON	-	JavaScript Object Notation
KIS	-	Krankenhausinformationssystem
MVCC	-	Multiversion Concurrency Control
PIN	-	Persönliche Identifikationsnummer
RDBMS	-	Relational Database Management System
RDF	-	Resource Description Framework
REST	-	Representational State Transfer
SQL	-	Strucured Query Language
YAML	-	Yet Another Markup Language

VI Anhang

Teil A: Implementierung des PVS - Prototypen

```
package pvs;

/**
 * @author Andy Krebs
 * Patientenverwaltungssystem
 */
public class PVS extends javax.swing.JFrame {

    public PVS() {
        initComponents();
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        PAnlegen pAnlegen = new PAnlegen();
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        PVerlegen pVerlegen = new PVerlegen();
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        PEntlassen pEntlassen = new PEntlassen();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /*
         * Set the Nimbus look and feel
         */
        Look and feel setting code (optional)

        /*
         * Create and display the form
         */
        java.awt.EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                new PVS().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    // End of variables declaration
}
```

Quellcode 3 Implementierung der PVS-Anwendung

```java
package pvs;

import java.io.IOException;

/**
 * @author Andy Krebs Frame zum Anlegen eines Patienten
 */
public class PAnlegen extends javax.swing.JFrame {

    private static Process p;

    public PAnlegen() {
        initComponents();
        setSize(500, 500);
        getContentPane();
        setVisible(true);
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        StringBuilder sb = new StringBuilder("{\\\"");

        String l1 = jLabel1.getText();
        String l2 = jLabel2.getText();
        String l3 = jLabel3.getText();
        String l4 = jLabel4.getText();
        String l5 = jLabel5.getText();
        String l6 = jLabel6.getText();
        String l7 = jLabel7.getText();

        String s1 = jTextField1.getText();
        String s2 = jTextField2.getText();
        String s3 = jTextField3.getText();
        String s4 = jTextField4.getText();
        String s5 = jTextField5.getText();
        String s6 = jTextField6.getText();
        String s7 = jTextField7.getText();

        String f1 = "\\\":\\\"";
        String f2 = "\\\",\\\"";
        sb = sb.append(l1).append(f1).append(s1).append(f2)
                .append(l2).append(f1).append(s2).append(f2)
                .append(l3).append(f1).append(s3).append(f2)
                .append(l4).append(f1).append(s4).append(f2)
                .append(l5).append(f1).append(s5).append(f2)
                .append(l6).append(f1).append(s6).append(f2)
                .append(l7).append(f1).append(s7).append("\\\"}");

        String s = sb.toString();


        String file = "c:/users/master/desktop/bachelor/curl/";
```

```
            String put = " -X PUT http://127.0.0.1:5984/patienten/" + s1
                    + " -d \"" + s + "\"";
            String command = "cmd /c start cmd.exe /C \"cd " + file + " && curl.exe
                    + put + ">data.txt -H \"Content-Type: application/json\"";
            try {
                setP(Runtime.getRuntime().exec(command));
            String put = " -X PUT http://127.0.0.1:5984/patienten/" + s1
                    + " -d \"" + s + "\"";
            String command = "cmd /c start cmd.exe /C \"cd " + file + " && curl.exe"
                    + put + ">data.txt -H \"Content-Type: application/json\"";
            try {
                setP(Runtime.getRuntime().exec(command));
                Thread.sleep(3000);
                getP().destroy();
            } catch (IOException | InterruptedException e) {
            }
        }

        /**
         * @param args the command line arguments
         */
        public static void main(String args[]) {...}

        public static Process getP() {...}

        public static void setP(Process p) {...}
        // Variables declaration - do not modify
        private javax.swing.JButton jButton1;
        private javax.swing.JLabel jLabel1;
        private javax.swing.JLabel jLabel2;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JLabel jLabel4;
        private javax.swing.JLabel jLabel5;
        private javax.swing.JLabel jLabel6;
        private javax.swing.JLabel jLabel7;
        private javax.swing.JLabel jLabel8;
        private javax.swing.JTextField jTextField1;
        private javax.swing.JTextField jTextField2;
        private javax.swing.JTextField jTextField3;
        private javax.swing.JTextField jTextField4;
        private javax.swing.JTextField jTextField5;
        private javax.swing.JTextField jTextField6;
        private javax.swing.JTextField jTextField7;
        // End of variables declaration
}
```

Quellcode 4 Implementierung für die Patientenaufnahme

```java
package pvs;

import ...

/**
 * @author Andy Krebs
 * Frame zum Verlegen eines Patienten
 */
public class PVerlegen extends javax.swing.JFrame {

    private static Process p;
    private static BufferedReader br;
    private static String rev, file;

    public PVerlegen() {
        initComponents();
        setSize(500, 500);
        getContentPane();
        setVisible(true);
        jButton2.setVisible(false);
        jLabel2.setVisible(false);
        jLabel3.setVisible(false);
        jTextField2.setVisible(false);
        jLabel4.setVisible(false);
        jTextField3.setVisible(false);
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        setFile("c:/users/master/desktop/bachelor/curl/");
        String get = "http://127.0.0.1:5984/patienten/" + s1;
        String command = "cmd /c start cmd.exe /C \"cd " + getFile()
                + " && curl.exe " + get + ">data.txt";
        try {
            setP(Runtime.getRuntime().exec(command));
            Thread.sleep(3000);
            getP().destroy();
        } catch (IOException | InterruptedException e) {
        }

        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();
            if (s.startsWith("{\"_id\":\"" + s1)) {
                jLabel3.setVisible(true);
                jLabel3.setText("Patient gefunden");
            } else {
                jLabel3.setText("Patient nicht vorhanden");
                jLabel3.setVisible(true);
            }

            StringBuilder sb = new StringBuilder(s);
```

```
            String arg0 = sb.substring(24);
            arg0 = arg0.substring(0, 3);
            if (arg0.endsWith("-")) {
                setRev(sb.substring(24, 59));
            } else {
                setRev(sb.substring(24, 58));
            }

            jButton2.setVisible(true);
            jLabel2.setVisible(true);
            jTextField2.setVisible(true);
            jLabel4.setVisible(true);
            jTextField3.setVisible(true);
        } catch (Exception e) {
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();

            StringBuilder sb = new StringBuilder(s);
            s = sb.substring(60);
            if (s.startsWith(",")) {
                s = sb.substring(61);
            }
            s = s.replace("\"", "\\\"");
            s = s.replace("}", "");

            String f1 = "\\\":\\\"";
            String f2 = "\\\",\\\"";
            String t2 = jTextField2.getText();
            String l2 = jLabel2.getText();
            String t3 = jTextField3.getText();
            String l3 = jLabel4.getText();
            StringBuilder sb2 = new StringBuilder(s);
            sb2 = sb2.append(",\\\"").append(l2).append(f1).append(t2)
                    .append(f2).append(l3).append(f1).append(t3).append("\\\"");
            String n = sb2.toString();
            String put = "-X PUT http://127.0.0.1:5984/patienten/" + s1
                    + "?rev=" + getRev() + " -d \"{" + n + "}\" ";
            String command = "cmd /c start cmd.exe /K \"cd " + getFile()
                    + " && curl.exe " + put
                    + ">data.txt -H \"Content-Type: application/json\"";
            try {
                setP(Runtime.getRuntime().exec(command));
                Thread.sleep(3000);
                getP().destroy();
            } catch (IOException | InterruptedException e) {
            }
        } catch (Exception e) {
        }
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {...}

    public static Process getP() {...}

    public static void setP(Process p) {...}

    public static BufferedReader getBr() {...}

    public static void setBr(BufferedReader br) {...}

    public static String getRev() {...}

    public static void setRev(String rev) {...}

    public static String getFile() {...}

    public static void setFile(String file) {...}
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    // End of variables declaration
}
```

Quellcode 5 Implementierung für die Patientenverlegung

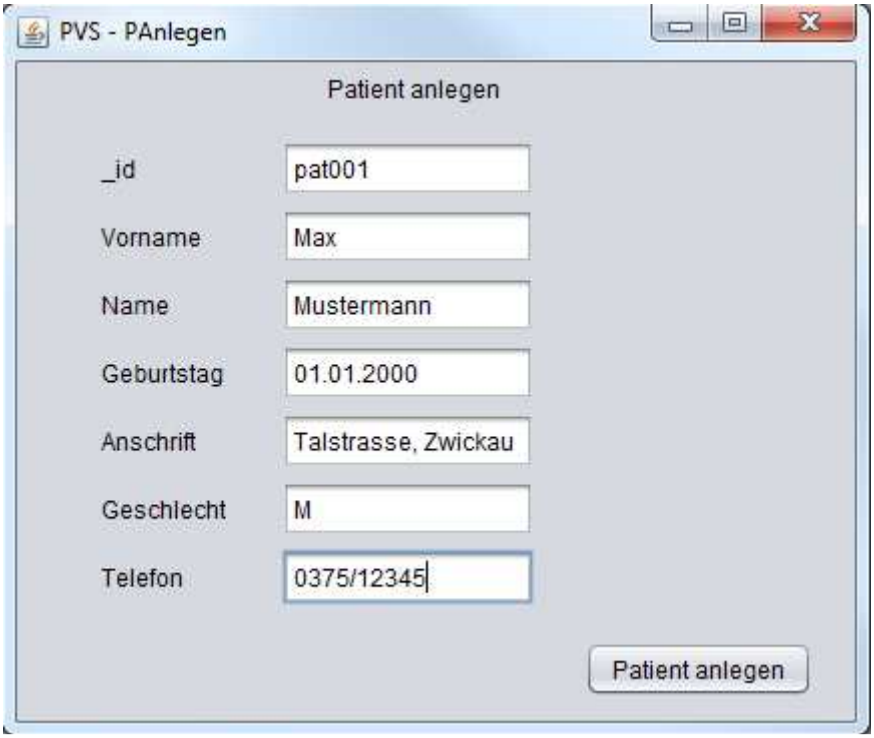

Abbildung 21 PVS - Patientenaufnahme

Abbildung 22 PVS - Patientenverlegung

Abbildung 23 PVS - Patientenentlassung

```java
package pvs;

import ...

/**
 * @author Andy Krebs
 * Frame zur Entlassung eines Patienten
 */
public class PEntlassen extends javax.swing.JFrame {

    private static Process p;
    private static BufferedReader br;
    private static String rev, file;

    public PEntlassen() {
        initComponents();
        setSize(500, 500);
        getContentPane();
        setVisible(true);
        jButton2.setVisible(false);
        jLabel3.setVisible(false);
        jLabel4.setVisible(false);
        jTextField3.setVisible(false);
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();

            StringBuilder sb = new StringBuilder(s);
            s = sb.substring(60);
            if (s.startsWith(",")) {
                s = sb.substring(61);
            }
            s = s.replace("\"", "\\\"");
            s = s.replace("}", "");

            String f1 = "\\\":\\\"";
            String f2 = "\\\",\\\"";
            String t2 = "entlassen";
            String l2 = "Status";
            String t3 = jTextField3.getText();
            String l3 = jLabel4.getText();

            StringBuilder sb2 = new StringBuilder(s);
            sb2 = sb2.append(",\\\"").append(l2).append(f1).append(t2)
                    .append(f2).append(l3).append(f1).append(t3).append("\\\"");
            String n = sb2.toString();
            String put = "-X PUT http://127.0.0.1:5984/patienten/" + s1
                    + "?rev=" + getRev() + " -d \"{" + n + "}\" ";
```

```
            String command = "cmd /c start cmd.exe /C \"cd " + getFile()
                    + " && curl.exe " + put
                    + ">data.txt -H \"Content-Type: application/json\"";
            try {
                setP(Runtime.getRuntime().exec(command));
                Thread.sleep(3000);
                getP().destroy();
            } catch (IOException | InterruptedException e) {
            }
        } catch (Exception e) {
        }
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        setFile("c:/users/master/desktop/bachelor/curl/");
        String get = "http://127.0.0.1:5984/patienten/" + s1;
        String command = "cmd /c start cmd.exe /C \"cd " + getFile()
                + " && curl.exe " + get + ">data.txt";
        try {
            setP(Runtime.getRuntime().exec(command));
            Thread.sleep(3000);
            getP().destroy();
        } catch (IOException | InterruptedException e) {
        }

        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();
            if (s.startsWith("{\"_id\":\"" + s1)) {
                jLabel3.setVisible(true);
                jLabel3.setText("Patient gefunden");
            } else {
                jLabel3.setText("Patient nicht vorhanden");
                jLabel3.setVisible(true);
            }

            StringBuilder sb = new StringBuilder(s);
            String arg0 = sb.substring(24);
            arg0 = arg0.substring(0, 3);
            if (arg0.endsWith("-")) {
                setRev(sb.substring(24, 59));
            } else {
                setRev(sb.substring(24, 58));
            }

            jButton2.setVisible(true);
            jLabel4.setVisible(true);
            jTextField3.setVisible(true);
        } catch (Exception e) {
        }
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {...}

    public static Process getP() {...}

    public static void setP(Process p) {...}

    public static BufferedReader getBr() {...}

    public static void setBr(BufferedReader br) {...}

    public static String getRev() {...}

    public static void setRev(String rev) {...}

    public static String getFile() {...}

    public static void setFile(String file) {...}
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField3;
    // End of variables declaration
}
```

Quellcode 6 Implementierung für die Patientenentlassung

Teil B: Implementierung des KDMS - Prototypen

```
package kdms;

/**
 * @author Andy Krebs
 * Klinisches Dokumentations- und Managementsystem
 */
public class KDMS extends javax.swing.JFrame {

    public KDMS() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        PDAnsehen pdAnsehen = new PDAnsehen();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {

        Look and feel setting code (optional)

        java.awt.EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                new KDMS().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration
}
```

Quellcode 7 Implementierung der KDMS-Anwendung

```java
package kdms;

import ...

/**
 * @author Andy Krebs
 * Frame zur Einsicht der Patientendaten
 */
public class PDAnsehen extends javax.swing.JFrame {

    private static Process p;
    private static String file, rev, get;
    private static BufferedReader br;

    public PDAnsehen() {
        initComponents();
        setSize(500, 500);
        getContentPane();
        setVisible(true);
        jLabel3.setVisible(false);
        jButton2.setVisible(false);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        setFile("c:/users/master/desktop/bachelor/curl/");
        get = ("http://127.0.0.1:5984/patienten/" + s1);
        String command = "cmd /c start cmd.exe /C \"cd " + getFile()
                + " && curl.exe " + get + ">data.txt";
        try {
            setP(Runtime.getRuntime().exec(command));
            Thread.sleep(3000);
            getP().destroy();
        } catch (IOException | InterruptedException e) {
        }

        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();
            if (s.startsWith("{\"_id\":\"" + s1)) {
                jLabel3.setVisible(true);
                jLabel3.setText("Patient gefunden");
            } else {
                jLabel3.setText("Patient nicht vorhanden");
                jLabel3.setVisible(true);
            }

            StringBuilder sb = new StringBuilder(s);
```

```
            String arg0 = sb.substring(24);
            arg0 = arg0.substring(0, 3);
            if (arg0.endsWith("-")) {
                setRev(sb.substring(24, 59));
            } else {
                setRev(sb.substring(24, 58));
            }
            jButton2.setVisible(true);

        } catch (Exception e) {
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            setBr(new BufferedReader(new FileReader(file + "data.txt")));
            String s = getBr().readLine();
            s = s.replace("\"\"", " ");
            s = s.replace("\",\"", "|");
            s = s.replace(",\"", "|");
            s = s.replace("\":\"", "|");
            s = s.replace("\":", "|");
            s = s.replace("{\"", "|");
            s = s.replace("\"}", "|");
            s = s.replace("||", "| |");
            s = s.replace("}}}", "|");
            s = s.replace("}", "|");

            //count der Trenner
            StringTokenizer stk = new StringTokenizer(s, "|");
            int count = 0;
            while (stk.hasMoreTokens()) {
                String token = stk.nextToken();
                count++;
            }

            int[] list = new int[count];
            int k = 0;
            int posOld = 0;
            String arg0 = "|";
            int t = s.lastIndexOf(arg0);
            for (int i = 0; i <= t; i++) {
                int pos = s.indexOf(arg0, i);
                if (pos != posOld) {
                    list[k] = pos;
                    k = k + 1;
                }
                posOld = pos;
            }

            String[][] rowData = new String[1][count / 2];
            String[] columnNames = new String[count / 2];

            columnNames[0] = s.substring(1, list[0]);
            rowData[0][0] = s.substring(list[0] + 1, list[1]);
```

```
            int j = 1;
            for (int i = 1; i < count / 2; i++) {
                columnNames[i] = s.substring(list[j] + 1, list[j + 1]);
                rowData[0][i] = s.substring(list[j + 1] + 1, list[j + 2]);
                j = j + 2;
            }
            JFrame f = new JFrame();
            f.setTitle("KDMS - Dateneinsicht");
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JTable table = new JTable(rowData, columnNames);
            f.add(new JScrollPane(table));
            f.pack();
            f.setVisible(true);
        } catch (IOException | HeadlessException e) {
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {...}

    public static Process getP() {...}

    public static void setP(Process p) {...}

    public static BufferedReader getBr() {...}

    public static void setBr(BufferedReader br) {...}

    public static String getFile() {...}

    public static void setFile(String file) {...}

    public static String getRev() {...}

    public static void setRev(String rev) {...}
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
}
```

Quellcode 8 Implementierung für die Dateneinsicht eines Patienten

Teil C: Implementierung des RIS - Prototypen

```
package ris;

/**
 * @author Andy Krebs
 * Radiologieinformationssystem
 */
public class RIS extends javax.swing.JFrame {

    public RIS() {
        initComponents();
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        RUntersuchung rUntersuchung = new RUntersuchung();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {...}
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration
}
```

Quellcode 9 Implementierung der RIS-Anwendung

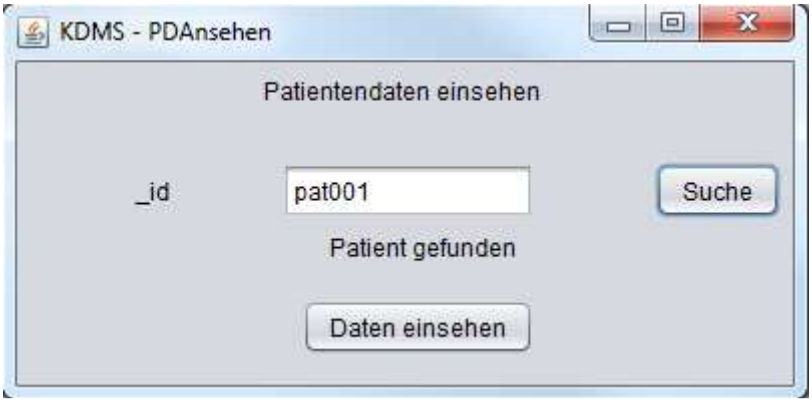

Abbildung 24 KDMS - Patientendaten einsehen

Abbildung 25 KDMS – Patientendaten einsehen – Dateneinsicht

```java
package ris;

import ...

/**
 * @author Andy Krebs
 * Frame zur radiologischen Untersuchung eines Patienten
 */
public class RUntersuchung extends javax.swing.JFrame {

    private static Process p, p2;
    private static BufferedReader br;
    private static String rev, file;

    public RUntersuchung() {
        initComponents();
        setSize(500, 500);
        getContentPane();
        setVisible(true);
        setUnvisible();
    }

    /**...*/
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        setFile("c:/users/master/desktop/bachelor/curl/");
        String get = "http://127.0.0.1:5984/patienten/" + s1;
        String command = "cmd /c start cmd.exe /C \"cd " + getFile()
                + " && curl.exe " + get + ">data.txt";
        try {
            setP(Runtime.getRuntime().exec(command));
            Thread.sleep(3000);
            getP().destroy();
        } catch (IOException | InterruptedException e) {
        }

        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String s = getBr().readLine();
            if (s.startsWith("{\"_id\":\"" + s1)) {
                jLabel3.setVisible(true);
                jLabel3.setText("Patient gefunden");
            } else {
                jLabel3.setText("Patient nicht vorhanden");
                jLabel3.setVisible(true);
            }

            StringBuilder sb = new StringBuilder(s);
            String arg0 = sb.substring(24);
            arg0 = arg0.substring(0, 3);
            if (arg0.endsWith("-")) {
                setRev(sb.substring(24, 59));
            } else {
```

```
                setRev(sb.substring(24, 58));
            }

            setVisibleAfterSearch();
        } catch (Exception e) {
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        StringBuilder sb = new StringBuilder("{\\\"");

        String l2 = jLabel2.getText();
        String l3 = jLabel4.getText();
        String l4 = jLabel5.getText();

        String s1 = jTextField1.getText();
        String s2 = jTextField2.getText();
        String s3 = jTextField3.getText();
        String s4 = jTextField4.getText();

        String f1 = "\\\":\\\"";
        String f2 = "\\\",\\\"";
        sb = sb.append(l2).append(f1).append(s2).append(f2)
                .append(l3).append(f1).append(s3).append(f2)
                .append(l4).append(f1).append(s4).append("\\\"}");

        String s = sb.toString();
        s = s.replace(",", ", ");
        s = s.replace("{", ", ");

        try {
            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String ausgabe = getBr().readLine();

            StringBuilder sb2 = new StringBuilder(ausgabe);
            ausgabe = sb2.substring(60);
            if (ausgabe.startsWith(",")) {
                ausgabe = sb2.substring(61);
            }
            ausgabe = ausgabe.replace("\"", "\\\"");
            ausgabe = ausgabe.replace(",", ", ");

            StringBuilder sb3 = new StringBuilder(ausgabe);
            sb3 = sb3.append(s);
            String n = sb3.toString();
            String put = "-X PUT http://127.0.0.1:5984/patienten/" + s1
                    + "?rev=" + getRev() + " -d \"{" + n + "\"";
            String command2 = "cmd /c start cmd.exe /C \"cd " + getFile()
                    + " && curl.exe " + put
                    + ">data.txt -H \"Content-Type: application/json\"";
            try {
                setP2(Runtime.getRuntime().exec(command2));
                Thread.sleep(3000);
                getP2().destroy();
                jLabel6.setVisible(true);
                jTextField5.setVisible(true);
```

```
                jButton3.setVisible(true);
            } catch (IOException | InterruptedException e) {
            }

            setBr(new BufferedReader(new FileReader(getFile() + "data.txt")));
            String aus = getBr().readLine();

            StringBuilder sb4 = new StringBuilder(aus);
            String arg0 = sb4.substring(32);
            arg0 = arg0.substring(0, 3);
            if (arg0.endsWith("-")) {
                setRev(sb4.substring(32, 67));
            } else {
                setRev(sb4.substring(32, 66));
            }

        } catch (Exception e) {
        }



    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        String s1 = jTextField1.getText();
        String upload = jTextField5.getText();

        String put = "-X PUT http://127.0.0.1:5984/patienten/" + s1 + "/"
                + upload + "?rev=" + getRev() + " --data-binary @" + upload;
        String command = "cmd /c start cmd.exe /K \"cd " + getFile()
                + " && curl.exe " + put + ">data.txt";
        try {
            setP(Runtime.getRuntime().exec(command));
            Thread.sleep(3000);
            getP().destroy();
        } catch (IOException | InterruptedException e) {
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {...}

    private void setUnvisible() {...}

    public void setVisibleAfterSearch() {...}

    public static Process getP() {...}

    public static void setP(Process p) {...}

    public static Process getP2() {...}

    public static void setP2(Process p) {...}
```

```
    public static BufferedReader getBr() {...}

    public static void setBr(BufferedReader br) {...}

    public static String getRev() {...}

    public static void setRev(String rev) {...}

    public static String getFile() {...}

    public static void setFile(String file) {...}
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    // End of variables declaration
}
```

Quellcode 10 Implementierung zum Anlegen einer radiologischen Untersuchung

Abbildung 26 RIS – Untersuchung eines Patienten anlegen

Legende für die Prozessketten:

Abbildung 27 Legende der Prozessketten

VII Selbstständigkeitserklärung gem. §22 Absatz 5 BPO

Hiermit versichere ich, Andy Krebs, dass ich vorliegende Bachelorarbeit mit dem Titel

„Möglichkeiten des Einsatzes von NoSQL-Technologien im Umfeld von Krankenhausinformationssystemen.“

Selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Werdau, den 13. August 2013 Unterschrift