

## INHALTSVERZEICHNIS

<b>1 Einleitung .....</b>	<b>3</b>
<b>1.1 Motivation.....</b>	<b>3</b>
<b>1.2 Vokabular.....</b>	<b>5</b>
<b>1.3 Anforderungen.....</b>	<b>6</b>
<b>2 Projektstruktur .....</b>	<b>8</b>
<b>2.1 Prototypen.....</b>	<b>8</b>
<b>2.2 Projektplan.....</b>	<b>9</b>
<b>2.3 Projektlayout.....</b>	<b>10</b>
2.3.1 Architektur.....	10
<b>3 Komponenten .....</b>	<b>13</b>
<b>3.1 SolrStack.....</b>	<b>13</b>
3.1.1 Apache Solr .....	13
3.1.2 Einrichtung .....	13
3.1.3 Grundaufbau Solr .....	16
3.1.4 Funktionalitäten .....	16
3.1.5 Schema-Datei .....	17
3.1.6 Konfigurationsdatei.....	30
<b>3.2 Clusterfähigkeit .....</b>	<b>44</b>
3.2.1 Solrcloud.....	44
3.2.2 Start des SolrCloud Clusters .....	45
3.2.3 ZooKeeper .....	49
<b>3.3 Webservice .....</b>	<b>50</b>
3.3.1 Anforderungen .....	50
3.3.2 Layout .....	52
3.3.3 Solr Klasse .....	53
3.3.4 REST-API.....	54
3.3.5 Weboberfläche .....	63
<b>3.4 Apache Nutch .....</b>	<b>68</b>
3.4.1 Version.....	68
3.4.2 Bestandteile.....	69
3.4.3 Funktionsweise .....	69
3.4.4 Bisherige Konfiguration .....	70
3.4.5 Was kann Nutch? .....	71
<b>4 Schluss .....</b>	<b>71</b>
<b>4.1 Zusammenfassung des Projektergebnisses.....</b>	<b>71</b>
<b>4.2 Ausblick .....</b>	<b>72</b>
4.2.1 Statistik.....	72
4.2.2 Sicherheit.....	73

## 1 EINLEITUNG

Der folgende Abschnitt gibt einen ersten Überblick über das dokumentierte Projekt, den Solr Suchservice. Dabei wird zuerst auf die Motivation für das Projekt eingegangen, gefolgt von einer Erklärung wichtiger Vokabeln. Im Anschluss werden die wesentlichen Anforderungen definiert.

### 1.1 MOTIVATION

Dieses Projekt beschäftigt sich mit dem Thema „Intelligente Suche“.

Suchmaschinen gehören mittlerweile zum Standardrepertoire von Internetnutzern. Fast jede Webseite enthält eine integrierte Suchmaschine. Somit erwarten die meisten Besucher mithilfe der Suche durch die Webseite navigieren zu können und zwar genauso effizient und effektiv wie sie es von kommerziellen Varianten, wie zum Beispiel „Google“, gewohnt sind. Diese kommerziellen, „mächtigen“ Suchen sind jedoch häufig sehr teuer.

Eine OpenSource Alternative dazu bietet Apache Solr, welche ebenso mächtig ist, wie kommerzielle Varianten. Solr ist allerdings schon in der Grundkonfiguration nicht trivial zu benutzen, sodass das Integrieren einer Suche in eine Webseite meistens einen hohen Aufwand erfordert. Das Ziel dieses Projektes ist es, Solr so zu konfigurieren, dass es sich einfach in einen bestehenden Internetauftritt einbauen lässt.

Das Projekt findet im Rahmen des Moduls Praxisprojekt Anwendungssysteme an der Technischen Universität statt und ermöglicht es Studenten Facetten realer Projektarbeit kennenzulernen (Meetings, Entscheidungsfindung, Projektdokumentation, Ergebnispräsentation, Implementierung und Umsetzung, etc.). Es fördert teamorientiert komplexe Aufgaben im Kontext der Gestaltung und Bewertung von IT-Systemen zu bewältigen. Im Folgenden sind die Projektbeteiligten aufgeführt:

#### **Das Fachgebiet Wirtschaftsinformatik – Information Systems Engineering (ISE) an der Fakultät IV der TU Berlin<sup>1</sup>**

Der Lehrstuhl erforscht die software-technische Gestaltung und interdisziplinäre Bewertung von verteilten, auf Plattformen basierten IT-Systemen in anspruchsvollen, zukunftsweisenden Anwendungsdomänen – innovativ, technikorientiert, mit Praxisbezug und mit dem Anspruch auf internationales Spitzenniveau. Das Fachgebiet unter der Leitung von Prof. Dr.-Ing. Stefan Tai wurde im Juni 2014 gegründet und befindet sich noch im Aufbau.

---

<sup>1</sup> <http://www.ise.tu-berlin.de/> (letzter Zugriff: 28.03.2015)

## 1.2 VOKABULAR

### Apache Nutch

Nutch<sup>3</sup> ist ein Webcrawler, der von Apache entwickelt wurde und zum Indizieren von Webseiten verwendet wird.

### API

API steht für *application programming interface* und ist die Funktionalität einer Software, die von anderen Programmen zur Kommunikation mit dieser Software genutzt werden kann.

### Caseing

Beim Caseing wird zwischen Groß- und Kleinschreibung unterschieden.

### Cluster

Unter Cluster versteht man eine Anzahl von Recheneinheiten, welche als Einheit arbeiten um Vorteile wie Fault-Tolerance, Loadbalancing und erhöhte Arbeitsgeschwindigkeit zu erreichen. Diese verteilten Systeme können durch virtuelle Maschinen auf einzelnen Rechnern laufen, werden im Allgemeinen aber durch Serverfarmen realisiert.

### Extracting

Das Extracting steht für den Vorgang eines Webcrawlers bestimmte Ausschnitte nach vordefinierten Regeln von Html-Dokumenten zu extrahieren.

### Fault-Tolerance und Loadbalancing

Fault-Tolerance beschreibt lediglich die Eigenschaft von Clustern, nicht bei jedem Fehler oder Absturz einer Instanz sofort jegliche Funktionalität zu verlieren.

Loadbalancing bedeutet im Zusammenhang mit diesem Projekt insbesondere, dass große Mengen an Suchanfragen gleichmäßig auf die Recheneinheiten verteilt werden, um Antwortzeiten zu verkürzen und Instanzen nicht zu überlasten.

### Facette / Facettierung

Durch individuelle Filterung der Suchergebnisse, die sogenannte Facettierung, finden Nutzer schneller zu den gesuchten Ergebnissen. Dies kann beispielsweise durch einen Dropdown oder eine Checkbox für Kategorien und Typen geschehen. Die Anwender können selbst dynamisch das Ergebnis beeinflussen, da jede Facette als Filter dient.

---

<sup>3</sup> <http://nutch.apache.org/> (letzter Zugriff: 14.3.2015)

## **Index**

Der Index ist eine kompakte Repräsentation aller für den Suchservice bereitgestellten Dateien. Neben Informationen über den Inhalt der Datei können auch unter anderen Daten bezüglich des Indizierungsdatums, der URL der Datei oder der Dateiname im Index gespeichert sein.

## **REST-API**

Eine REST-API ist eine API, welche nach einem ganz bestimmten Programmierparadigma für verteilte Systeme gestaltet ist und HTTP-Anfragen zur Kommunikation nutzt. Dieses sieht vor, dass eine Web-Adresse genau einen Seiteninhalt repräsentiert.

## **Suggestion**

Dies bezeichnet die Autovervollständigung von Benutzereingaben. Die möglichen Suggestions werden meist auf Grundlage eines Wörterbuches gebildet.

## **Stemming**

Bezeichnet ein Verfahren, mit dem verschiedene morphologische Varianten eines Wortes auf ihren gemeinsamen Wortstamm zurückgeführt werden können.

## **Webcrawler**

Ein Webcrawler ist ein automatisiertes Programm, welches das World Wide Web durchsucht und die Inhalte und Metadaten von Webseiten analysiert.

## **Webservice**

Ein Webservice ist eine Anwendung, welche über ein Netzwerk beispielsweise mithilfe eines Browsers wie Firefox<sup>4</sup> benutzt werden kann.

### **1.3 ANFORDERUNGEN**

Das Ziel des Projektes ist es, einen Suchservice auf Basis von Apache Solr zu entwickeln. Bei der Projektvorstellung zu Beginn des Semesters und beim offiziellen Kick-off wurde die Projektaufgabe und die zu erfüllenden Anforderungen vorgestellt. Der Suchservice soll autark vom Webauftritt lauffähig sein, in welchen die Suchmaschine später integriert werden soll. Das bedeutet, dass ein generelles und globales Konzept entwickelt werden muss. Die Suche soll über ein einziges Eingabefeld, wie es beispielsweise bei der Google-Suche zu finden ist und das in eine Webseite eingebaut werden kann, funktionieren. Das Projekt sollte laut Anforderungsvorgaben praxisnah gestaltet werden und ist deshalb als Festpreis-Projekt "ausgeschrieben": Es gibt eine feste Anzahl von Studenten, die in einem fixen Zeitrahmen die Anforderungen umsetzen müssen.

Die fertig umgesetzte Suchmaschine soll als Cluster mittels Virtueller Maschinen (VM) laufen können. Mit mindestens 2 VMs kann eine gewisse Ausfallsicherheit gewährleistet

---

<sup>4</sup> <https://www.mozilla.org/de/firefox/new/>

werden. Die Suchplattform soll auf Basis von Apache Solr aufgebaut sein und es soll eine sinnvolle Grundkonfiguration, die unter anderem Stemming und Caseing enthält, entwickelt werden. Weiterhin soll eine abstrahierende und vereinfachte API erstellt werden, welche alle sinnvollen Einstellungen einmalig umfasst. Sie soll als REST-Service in Java implementiert werden. Dabei übernimmt die REST-API verschiedene Funktionen, so soll sie die globalen Suchdaten mit Inhalten zur Indizierung an die Grundkonfiguration von Solr übergeben und Suchanfragen mithilfe von Solr beantworten. Zudem kann sie Facettengruppen deklarieren und automatisch die Vervollständigung von Suchbegriffen abfragen. Die Facettengruppen müssen bei der Suche kombinierbar sein und sollen über die REST-API hinzugefügt und entfernt werden können. Damit der Suchservice einfach anzuwenden ist, soll eine eigenständige globale Suchseite zur Nutzung der Suche konstruiert werden. Diese soll ein „zeitloses Design“ aufweisen und leicht in vorhandene Webseiten einzubauen sein. Die globale Suchseite darf nur über die neue REST-API Suchergebnisse erhalten.

Des Weiteren sollte es ohne Probleme möglich sein initial den Suchindex aufzubauen, beispielsweise mithilfe eines Webcrawlers.

Eine weitere Anforderung ist es, im Laufe des Projektes die Zwischenergebnisse zu präsentieren und eine Abschlussdemonstration vorzuführen. Bei dieser soll die Suchmaschine die Inhalte der TU Website in durchsuchbarer Form enthalten. Die Facetten sollen auf die verschiedenen Navigationspunkte der Webseite angepasst werden.

Die bisher erwähnten Anforderungen wurden erfolgreich realisiert. Eine mögliche Erweiterung der Suchmaschine sind Statistikwerkzeuge, damit eine Auswertung und Analyse von Suchanfragen vorgenommen und Suchanfragen anderer Benutzer genutzt werden können. Mit diesen Hilfsmitteln wäre es unter anderem möglich dem Anwender Informationen darüber zu geben was am meisten gesucht wurde, nach welchen Inhalten andere Besucher gesucht haben oder die Top 100 Suchanfragen des letzten Monats. Auf diesen Anwendungsfall wird im Abschnitt 4.2 eingegangen. Auch spielt Sicherheit bei Webanwendungen eine bedeutende Rolle, wie mit Sicherheitsaspekten umgegangen werden kann, findet sich im Abschnitt 4.2.2.

Zusätzlich zu den gegebenen Anforderungen mussten Rahmenbedingungen beachtet werden. Eine Einschränkung ist es, dass die Website der TU-Berlin nicht an den Suchservice angepasst werden kann oder muss. Zudem sollte ein Grabber geschrieben werden, welcher die Inhalte der Webseite periodisch abzieht und die ausgelesenen Inhalte über die neu erstellte API in der Suchmaschine indiziert. Dadurch entsteht die Möglichkeit zu Beginn schnell einen Suchindex aufzubauen. Die ausgelesenen Suchinhalte sollten weiterhin auf die TU-Berlin Webseite verlinken und die globale Suchseite soll über einen externen Link aufgerufen werden.

## 2 PROJEKTSTRUKTUR

Im Folgenden werden relevante Themen für den Projektablauf erklärt. Dabei findet ein kurzer Exkurs in die geplanten Prototypen des Suchservices, in den Projektplan und in die Projektstruktur statt, welche die einzelnen Komponenten näher beleuchtet.

### 2.1 PROTOTYPEN

Um die in Abschnitt 1.3 genannten Anforderungen zu erfüllen, wurden sie in verschiedene Arbeitspakete, bezeichnet als Prototypen, aufgeteilt. Sie stellen Meilensteine im Ablauf dar, welche zu einem gewissen Grad eine Erfüllung der Anforderungen aufzeigen. Ebenfalls erleichtert die Aufteilung in Prototypen die Präsentation von Teilen der Arbeit innerhalb des Projekts. Die Namen der Prototypen sind dabei Muss-Prototyp, bestehend aus Muss 1 und Muss 2 und Kann-Prototyp. Erstere beinhalten folglich Anforderungen, welche eindeutig erfüllt sein sollten, wohingegen der letzte weitere Neuerungen beinhaltet, die nicht zu den Hauptbedingungen gehören.

#### Muss-Prototyp

Der Muss-Prototyp vereint die Prototypen Muss 1 und Muss 2, welche im Folgenden genauer beschrieben werden.

##### Muss 1

Der erste Prototyp, welcher als Ansatz für erste Vorführungen gilt, ist ein Solr Sever, welcher in einem verteilten System läuft und dabei Informationen durch einen Grabber erhält. Dabei ist bereits eine Webseite implementiert, durch die der Server ansprechbar ist.

Als Grundbaustein dafür ist die Beispiel-Suche von Solr angesetzt, um zunächst grobe Übersichten über das Gesamtprojekt zu besitzen. Im Vordergrund stehen hier also die Recherche und Einarbeitung in die einzelnen Themengebiete.

##### Muss 2

Der zweite Muss-Prototyp stellt bereits eine fertige Version des Suchservices dar, welche alle Hauptanforderungen erfüllt. Neben den in Muss 1 genannten Funktionen ist dabei die Integration von Facetten der Fokus dieser Version. So besteht die Möglichkeit Facetten über die Webseite einzustellen. Der Grabber muss es dabei schaffen, einen Index zu übertragen, welche diese Funktion zulässt und unter Umständen vereinfacht.

Der Solr Server wird außerdem mit einer anderen Cluster-Version verwendet, welche auf eigenständigen ZooKeeper-Instanzen läuft.

#### Kann-Prototyp

Anforderung an die letzte Version der Arbeit ist insbesondere die Unterstützung von Suchvorschlägen. Des Weiteren beherrscht dieser letzte Prototyp die Möglichkeit manuell Dokumente zum Index der Suchmaschine hinzuzufügen und zu entfernen. Dabei sollen die meisten gängigen Dokumenttypen in den Index geladen werden können. Alle vorherigen

Ergebnisse werden in diesem Release natürlich nicht verworfen, so dass dieser Prototyp eine Abarbeitung aller Anforderungen darstellt, die vom Team als durchführbar erachtet wurden.

Diese Definition des Prototyps ist eine neuere Variante, da im Laufe des Projektes Änderungen vorgenommen werden mussten. Der alte Kann-Prototyp hatte ausschließlich die Unterstützung von Suchvorschlägen und die Analyse von Suchergebnissen als statistisches Hilfsmittel zum Ziel, welche nun jedoch nicht mehr zum Kann-Prototyp gehört.

## 2.2 PROJEKTPLAN

Das Projekt wurde am 29.10.2014 in Form eines Kick-Offs bei adesso gestartet. Nach einer 2-tägigen Projektplanung begann danach die Recherche zu den zuvor zugewiesenen Themengebieten. Die Themengebiete teilten sich auf in der Grundkonfiguration von Solr, das Erstellen eines über eine Rest-API anzusprechenden Webservices, das Bereitstellen der Clusterfähigkeit und der Integration des Webcrawlers Apache Nutch. Anschließend konnte bereits mit der technischen Umsetzung der Prototypen begonnen werden, wobei mit den beiden Muss-Prototypen begonnen wurde, da erst im Anschluss der Kann-Prototyp kommt. Die Bearbeitung der Muss-Prototypen begann am 10.11.2014 und wurde mit jeweils einer Bearbeitungsdauer von einem Monat eingeplant. Die Umsetzung bestand jeweils aus den Arbeitspaketen Recherche, Konzeption, Umsetzung, Test und Release (siehe Abbildung 2-1: Projektplan (1)). Nach den Umsetzungen erfolgte am 17.12.2014 die Zwischenpräsentation der Zwischenergebnisse bei Adesso.

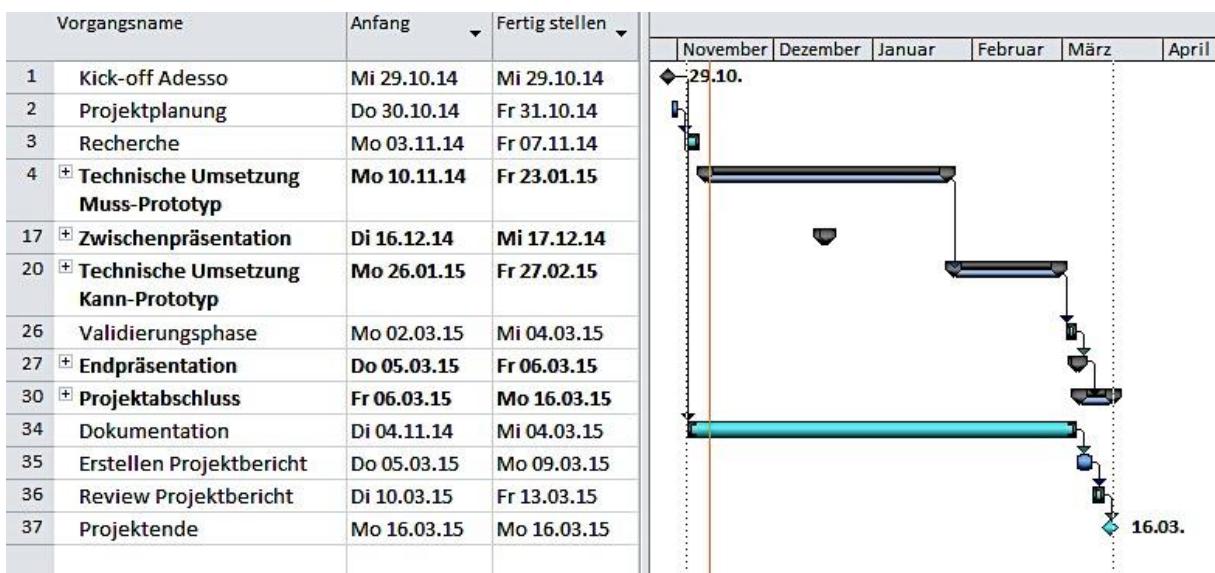


Abbildung 2-1: Projektplan (1)

Anschließend wurde die Umsetzung des Kann-Prototypen begonnen, für den ebenso eine 1-monatige Dauer angesetzt war. Die Arbeitspakete beim Kann-Prototyp wurden ebenfalls wie beim Muss-Prototyp bearbeitet. Im Anschluss war eine 3 Tage lange Validierungsphase eingeplant, welche am 6.3.2015 in der Endpräsentation bei Adesso mündete. Im Anschluss folgte der Projektabschluss verbunden mit der Abschlussdokumentation und Berichterstellung.

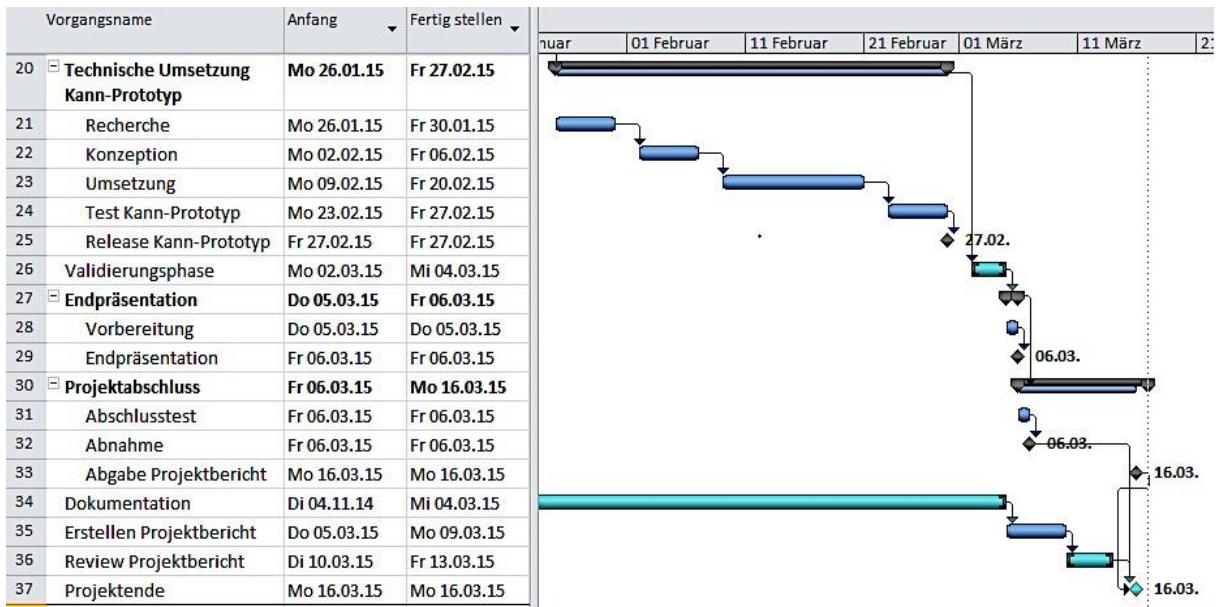


Abbildung 2-2: Projektplan (2)

## 2.3 PROJEKTLAYOUT

Wie in den Prototypen schon aufgezeigt, besteht das Projekt aus mehreren eigenständigen Komponentengruppen, welche im Zusammenspiel den Suchservice ergeben. Im folgenden Abschnitt werden diese kurz erklärt sowie wichtige Programmabläufe dargestellt.

### 2.3.1 ARCHITEKTUR

Der Suchservice beinhaltet folgende 3 Komponentengruppen.

#### 1. SolrStack und Cluster

Der SolrStack basiert auf Apache Solr<sup>5</sup> und ist praktisch die „Suchmaschine“. Hier wird der Index verwaltet und Suchanfragen werden bearbeitet. Der SolrStack kann entweder lokal auf einer Maschine ausgeführt werden oder als verteiltes Cluster laufen. Der Zugriff auf ihn ändert sich dadurch jedoch nicht, weswegen im weiteren Verlauf dieses Abschnittes nur die Bezeichnung SolrStack verwendet wird.

#### 2. Webservice und Rest-API

Der Webservice stellt eine REST-API zur Verfügung, über welche unter anderem die Beispielimplementierung zur Nutzung des Suchservices (die Weboberfläche, siehe 3.3.5 Weboberfläche) oder andere Anwendungen einen Zugriff mittels HTTP-Anfragen erhalten. Die Zugriffe werden im Webservice aufbereitet, welcher mit dem SolrStack kommuniziert, um die erhaltenen Anfragen umzusetzen.

<sup>5</sup> <https://lucene.apache.org/solr/>

### 3. Nutch

Neben dem Zugriff auf dem SolrStack über den Webservice, können Daten auch mit Hilfe des Webcrawlers Nutch vollautomatisch von zuvor ausgewählten Webseiten indiziert werden.

#### Zusammenspiel der Komponentengruppen

In Abbildung 2-3 ist das Zusammenspiel der Komponentengruppen veranschaulicht. Der SolrStack kommuniziert mit dem Webservice und wird zusätzlich durch Nutch mit Informationen versorgt. Anwendungen und die *Webseite* können den Suchservice über die REST-API Nutzen, welche durch den Webservice bereitgestellt wird.

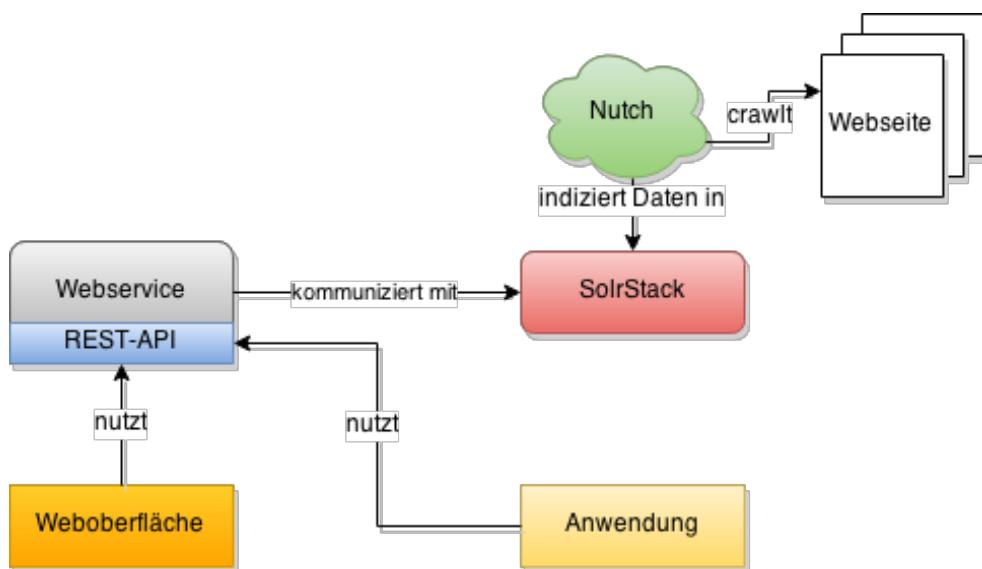


Abbildung 2-3: Zusammenspiel der Komponenten

Nähere Details zu den einzelnen Komponenten sind im Abschnitt 3 Komponenten aufgeführt.

Im folgenden Abschnitt sind die wichtigsten Programmabläufe des Suchservices dargestellt. In Abbildung 2-4 ist der sequentiellen Ablauf einer Standardsuchanfrage zu sehen. Zunächst stellt der Anfragesteller die Suchanfrage, welche die REST-API anspricht. Die REST-API sorgt dafür, dass die Anfrage über die Solr Klasse (siehe 3.3.3 Solr Klasse) im Webservice an den SolrStack gelangt. Im nächsten Schritt liefert der SolrStack die Antwort auf die Suchanfrage und einige Zusatzinformationen (wie z.B. die Anzahl der Suchergebnisse) über den gleichen Weg zurück, d.h. über die Solr Klasse über die REST-API zurück an den Anfragesteller.

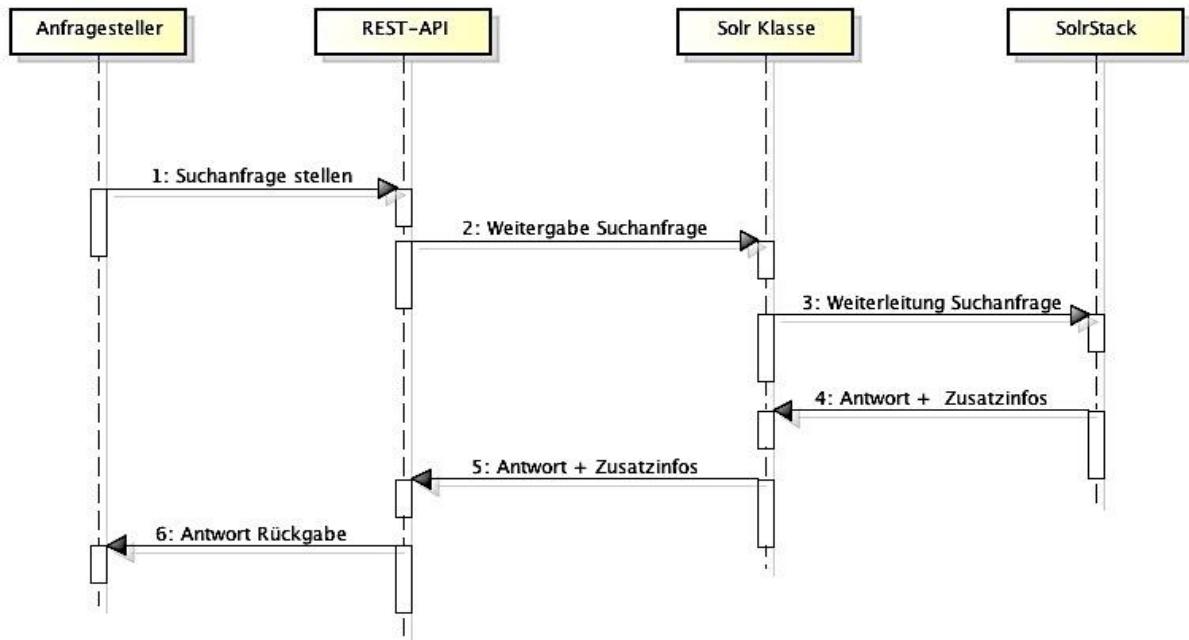


Abbildung 2-4: Standardsuchanfrage

In Abbildung 2-5 sieht man wie Nutch alle Informationen aus einer spezifizierten Webseite crawlt und sie anschließend auf relevanten Inhalt filtert. Sobald die Verarbeitung beendet ist, werden die Informationen in den SolrStack indiziert und können somit für die oben erwähnten Suchanfragen genutzt werden.

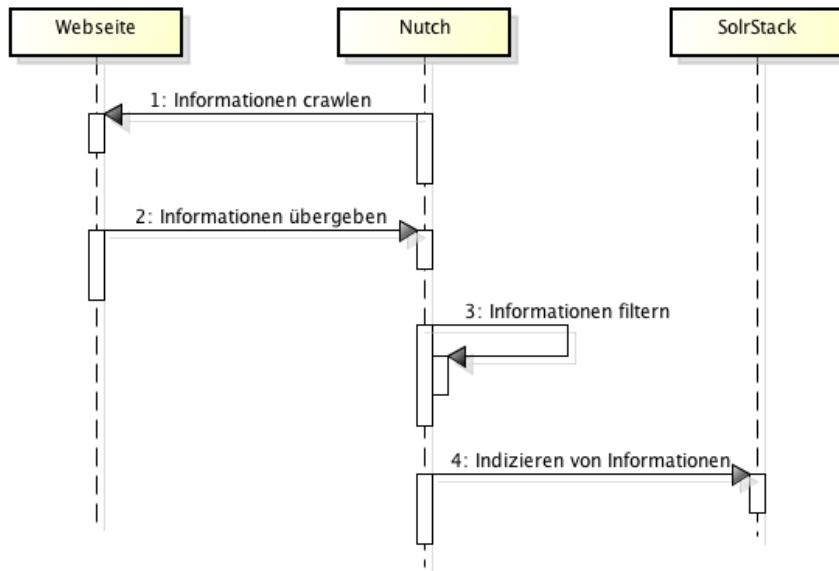


Abbildung 2-5: Informationsversorgung

Weitere Abläufe, insbesondere von die durch Nutzeranfragen über die REST-API ausgelösten Abläufe, sind im Abschnitt 3.3.4 aufgeführt und erläutert.

## 3 KOMPONENTEN

Im folgenden Abschnitt werden die vorher schon aufgezeigten Komponenten nochmals im Detail betrachtet. Dabei werden auch grundlegende Informationen und Details zur Herangehensweise sowie Probleme und Lösungen bezüglich der Komponenten dokumentiert.

### 3.1 SOLRSTACK

Der SolrStack ist die an die Anforderungen angepasste Konfiguration von Solr. Er ist die eigentliche Suchmaschine im Hintergrund, welche die Suchanfragen bearbeitet und die gespeicherten Inhalte im Index verwaltet. Der SolrStack basiert auf Apache Solr und kann entweder im Cluster oder Lokal ausgeführt werden.

Wie bereits im Abschnitt 2.1 Prototypen aufgezeigt, wurden die ersten Anforderungen an den SolrStack unter dem Begriff Grundkonfiguration zusammengefasst. In der fertigen Grundkonfiguration sind Stemming, Caseing, StopWords, Synonyme und Umlaute enthalten. Des Weiteren wurden die Funktionalitäten Suggetions und Facetten realisiert. Wie dies umgesetzt wurde, wird in den folgenden Abschnitten erläutert, insbesondere in 3.1.5 Schema-Datei und in 3.1.6 Konfigurationsdatei. Informationen darüber wie der SolrStack gestartet werden kann, finden sich im Abschnitt 3.1.2 Einrichtung oder im Handbuch.

#### 3.1.1 APACHE SOLR

Apache Solr wurde 2004 als eigenständige Suchplattform für CNET Networks-Intranet entwickelt und ist seit 2006/2007 als Open-Source Suchplattform verfügbar. 2011 wurden Lucene und Solr zu einer einheitlichen Versionierung miteinander verbunden. Die aktuellste Version ist Solr 5.0.<sup>6</sup>

Solr funktioniert als isolierte Suchmaschine für Volltext-Suchen. Es kann für Texte oder auch strukturiertere Dokumenten verwendet werden.<sup>7</sup>

Zudem verwendet Solr die Lucene Search Engine Library und erweitert hierdurch die Funktionalitäten, die in Abschnitt 3.1.4 weiter erläutert werden. Durch den Einbau von REST-Schnittstellen und XML-basierten Konfiguration können Suchergebnisse an individuelle Wünsche angepasst werden.<sup>8</sup>

#### 3.1.2 EINRICHTUNG

Beim fertigen Suchservice ist eine funktionsfähige Solr-Konfiguration bereits integriert, welche lediglich mit einem Befehl gestartet werden muss. Dafür muss zunächst in der Konsole in den SolrStack Ordner navigiert werden. In diesem wird dann der Befehl "java -jar start.jar". ausgeführt. Anschließend ist der SolrServer über ",,http://localhost:8983/solr/" ansprechbar.

<sup>6</sup> <http://www.apache.org/dyn/closer.cgi/lucene/solr/5.0.0> (letzter Zugriff: 28.03.2015)

<sup>7</sup> <http://spinfowiki.uni-koeln.de/lehre/images/d/d8/2011-12-12-Apache-Solr.pdf> (letzter Zugriff: 28.03.2015)

<sup>8</sup> <http://www.shi-gmbh.com/search-and-big-data/produkte/apache-solr> (letzter Zugriff: 28.03.2015)

Möchte man seine eigene Solr-Konfiguration ausarbeiten, muss zunächst Solr installiert werden. Die Installation von Solr erfolgt in sieben Schritten.

1. Zunächst sollte vor einer Installation sichergestellt werden, dass die aktuelle Java Version auf 1.7 oder höher ist.
2. Die Installationsdatei für den weiteren Verlauf sollte passend zum Betriebssystem gewählt werden. In Abbildung 3-1 sieht man die Auswahl an Installationsdateien: Für Linux, Unix und OSX steht die Datei mit der Endung „tgz“ zur Verfügung und Windows die Datei mit „zip“ als Endung benötigt.

Name	Last modified	Size	Description
<a href="#">Parent Directory</a>		-	
<a href="#">changes/</a>	2014-10-29 14:33	-	
<a href="#">solr-4.10.2-src.tgz</a>	2014-10-29 14:30	34M	
<a href="#">solr-4.10.2.tgz</a>	2014-10-29 14:30	143M	
<a href="#">solr-4.10.2.zip</a>	2014-10-29 14:30	149M	

Apache/2.4.7 (Ubuntu) Server at mirror.serversupportforum.de Port 80

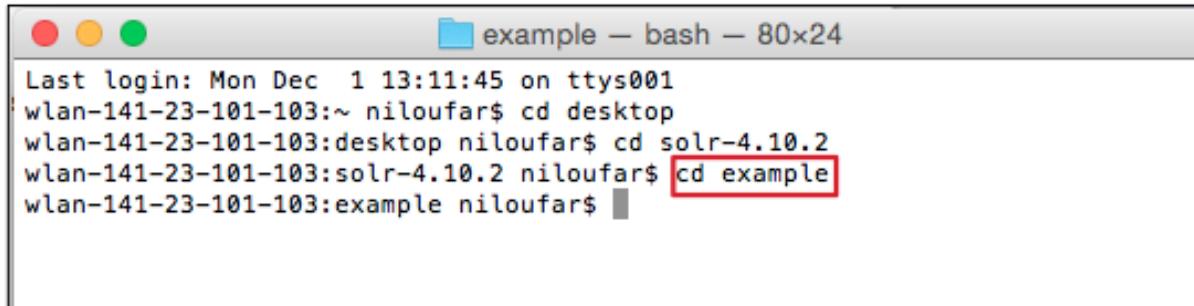
Abbildung 3-1: Installationsdatei

3. Nach dem Download sollte die Datei entweder im Terminal mit Hilfe von passenden Befehlen oder manuell entpackt werden und der Speicherort festgelegt werden.
4. Nach dem Entpacken sollte weiter im Terminal gearbeitet werden. Hier muss zunächst mit dem Befehl „cd“ und dem entsprechenden Speicherort zum Verzeichnis gewechselt werden und danach mit dem Befehl „cd solr-4.10.2“ auf den Ordner für die Installation zugegriffen werden, wobei „cd“ der Befehl für den Wechsel in ein anderes Verzeichnis steht und mit dem Zusatz „solr-4.10.2“ zum benötigten Ordner wechselt. Die beschriebenen Befehle sind in Abbildung 3-2 mit entsprechenden Markierungen sichtbar.

```
solr-4.10.2 - bash - 80x24
Last login: Mon Dec  1 13:11:45 on ttys001
wlan-141-23-101-103:~ niloufar$ cd desktop
wlan-141-23-101-103:desktop niloufar$ cd solr-4.10.2
wlan-141-23-101-103:solr-4.10.2 niloufar$
```

Abbildung 3-2: Verzeichniswechsel Solr-Ordner

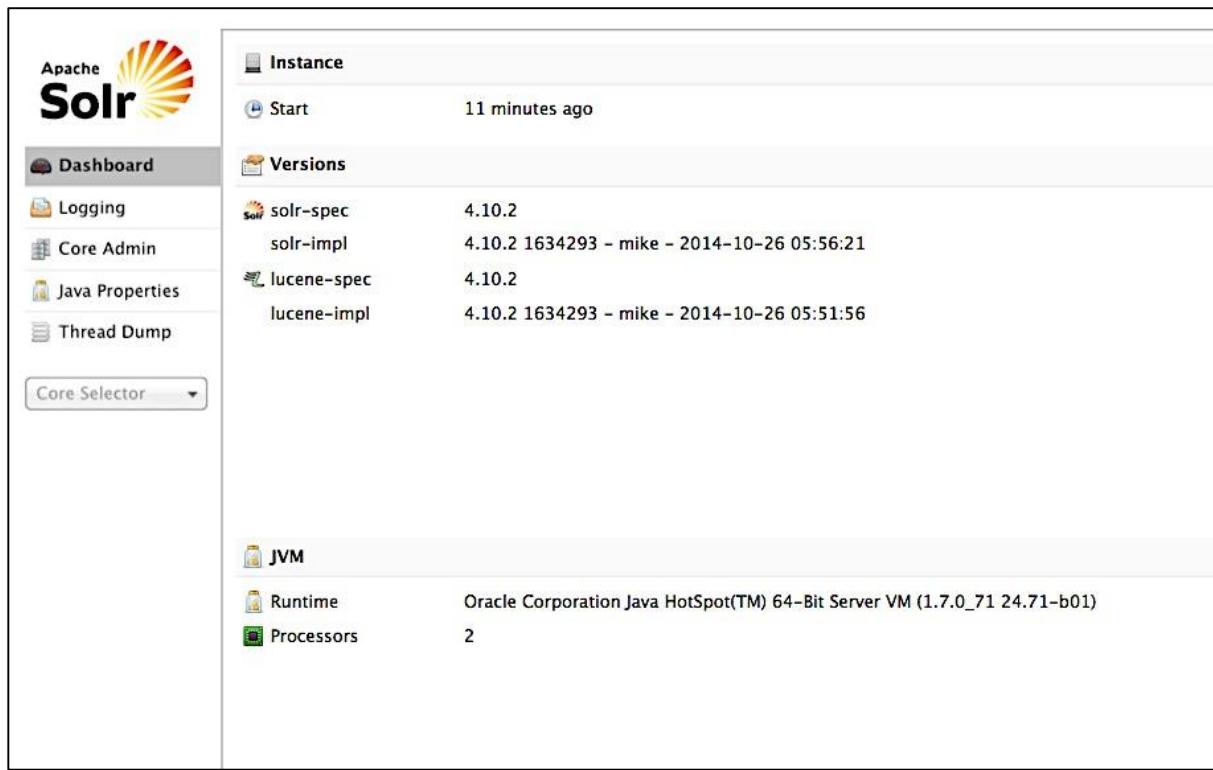
5. Weiterhin wird im Terminal nun in den Ordner „example“ gewechselt, was entsprechend in Abbildung 3-3 sichtbar ist. Hierfür wird nochmals der Befehl „cd“ und der entsprechende Ordner-Name verwendet.



```
Last login: Mon Dec  1 13:11:45 on ttys001
wlan-141-23-101-103:~ niloufar$ cd desktop
wlan-141-23-101-103:desktop niloufar$ cd solr-4.10.2
wlan-141-23-101-103:solr-4.10.2 niloufar$ cd example
wlan-141-23-101-103:example niloufar$
```

Abbildung 3-3: Verzeichniswechsel example-Datei

6. Um den Solr-Server nun zu starten, muss im Terminal der Befehl „java -jar start.jar“ ausgeführt werden. Der Server ist gestartet und kann durch das Aufrufen der Seite „<http://localhost:8983/solr/>“ in einem Internet-Browser überprüft werden.  
 7. Wenn alles richtig ausgeführt wurde, sieht man die geöffnete Solr-Seite, wie in Abbildung 3-4 zu sehen ist. Ist die Installation abgeschlossen kann mit dieser Solrdatei gearbeitet werden.



Instance	
	Start 11 minutes ago
Versions	
	solr-spec 4.10.2
	solr-impl 4.10.2 1634293 - mike - 2014-10-26 05:56:21
	lucene-spec 4.10.2
	lucene-impl 4.10.2 1634293 - mike - 2014-10-26 05:51:56
JVM	
	Runtime Oracle Corporation Java HotSpot(TM) 64-Bit Server VM (1.7.0_71 24.71-b01)
	Processors 2

Abbildung 3-4: Solr-Server

### 3.1.3 GRUNDAUFBAU SOLR

Solr baut auf mehrere Komponenten auf. Es besteht unter anderem aus der Schema-Datei (schema.xml) und der Konfigurationsdatei (solr.config).

Der genauere Aufbau der schema-Datei wird im Abschnitt 3.1.5 näher erläutert. Dieser Teil beinhaltet sowohl die Erläuterungen zu den Solr-Felder (3.1.5.1) als auch zu den Solr-Feldtypen (3.1.5.2). Der Abschnitt 3.1.6 geht näher auf die Konfigurationsdatei solr.config ein und erklärt die zugehörigen Befehle, die genutzt wurden.

Innerhalb des Suchservers können Inhalte gespeichert und abgefragt werden. Die Speicherung erfolgt in einem Index durch sogenannte Indexierung. Der Server kann ohne Probleme über HTTP-GET angesprochen werden, da es nicht an eine bestimmte Programmiersprache gebunden ist. Der Befehl muss an einen RequestHandler gerichtet sein. Folglich wird eine Suchanfrage an den SearchHandler gerichtet. Die Indexierung erfolgt durch den UpdateRequestHandler.<sup>9</sup>

#### 3.1.3.1 AUFBAU EINES SOLRDOKUMENTS

Ein Dokument welches in den Suchservice indiziert werden kann, kann verschieden aufgebaut sein. Einige Felder sind jedoch erforderlich:

**id:** damit jeder indizierte Inhalt eindeutig identifiziert werden kann, in diesem Suchservice st dies meist der Titel der Datei

**url:** gibt die URL zu der Datei an, z.B. die Website über die die Ursprungsdatei erreichbar ist

**content:** der eigentliche Inhalt des Dokumentes , dies ist auch das Feld in dem nach den Suchbegriffen gesucht wird

**title:** der Titel des Dokumentes

**tstamp:** gibt den Zeitpunkt der Indexierung an

**cat:** Facetten des Dokumentes

### 3.1.4 FUNKTIONALITÄTEN

Zuvor wurde inhaltlich auf den Grundaufbau von Solr eingegangen. Nun werden die einzelnen Funktionalitäten, die Solr anbietet, näher erläutert.

Solr ist zur Indizierung und Abfrage von digitalen Informationen sehr gut geeignet. Dafür bringt Solr viele Funktionen mit, von denen einige in den folgenden Absätzen aufgezählt werden. Der Server besitzt eine Volltextsuche, eine automatische Vervollständigung bei der Eingabe von Suchbegriffen und kann Suchergebnisse gewichten. Die Suche kann auch durch eine Facettierung durch einen Filter, Rechtschreibprüfung und Stemming (Stammworterkennung) erweitert werden. Durch weitere Einstellungen können auch Suchergebnisse angezeigt werden, in denen auch passende Synonyme erscheinen können.

<sup>9</sup> Klose, M., & Wrigley, D. (2014). *Einführung in Apache Solr*. O'Reilly Germany

Zusätzlich kann durch Vorschläge für ähnliche Treffer durch den Zusatz „Meinten Sie...?“ die Suche erweitert werden.<sup>10</sup>

Apache Solr ist auch durch Plugins beliebig erweiterbar und kann auch durch bestimmte Schnittstellen (z.B. über die REST-API) zur Eingabe bzw. Ausgabe von Daten angesprochen werden. Die benötigte Konfiguration erfolgt durch XML-Dateien.<sup>11</sup>

### 3.1.5 SCHEMA-DATEI

Die Schema.xml liegt im conf-Verzeichnis von Solr und dient zur Konfiguration des Index. Hier wird bestimmt, welche Felder im Index existieren und wie diese verarbeitet werden.

Im Allgemeinen kann man sagen, dass die schema.xml in drei Teile eingeteilt ist. In diesen Teilen werden die Felder, Feldtypen und allgemeine Einstellungen definiert.

#### 3.1.5.1 SOLR-FELDER

In dem XML-Tag <fields> werden die Felder eines Index definiert. Jedes in Solr abgelegte Dokument setzt sich zusammen aus Feldern (Eingangsdatum, Überschrift, Volltext, etc). Ein Dokument kann daher auch als Datensatz betrachtet werden.

Jedes Feld ist dabei zu definieren. Der Aufbau von Solr-Feldern ist identisch zu dem Aufbau von Feld-Typen, außer dass hier *name* und *type* die einzigen Pflichtproperties sind. Alle anderen Properties können das Default- Verhalten vom Feldtypen erweitern.

- *name*:  
Wichtig ist der eindeutige Name des Felds. und die Groß- und Kleinschreibung. Leerzeichen sollten eher vermieden werden, um diese bei der Suche nicht umgehen zu müssen.
- *type*:  
Hier wird der Typ des Felds definiert, welcher in der Typensektion der schema.xml vorhanden sein muss, damit es nicht zu einer Parse Exception beim Start von Solr kommt.
- *default*:  
Durch dieses Feld wird ein Default-Wert spezifiziert, falls dieser noch nicht bei der Indexierung kein Wert vorhanden ist. Dies hilft, um bei Trefferlisten Felder anzuzeigen, die keinen Wert haben.

Indexed und stored sind Parameter, die getrennt behandelt werden, was bedeutet, dass ein Element (Feld) indiziert sein kann, aber nicht gespeichert sein muss. Dieser Aspekt ist jedoch nur von Bedeutung, wenn Solr als Volltextindex genutzt werden soll, wobei dann eine eigene Datenbank für die Daten existieren muss. Bei diesem Fall muss auch eine große Datenmenge

<sup>10</sup> <http://www.zweiblog.com/2012/07/solr-die-hochperformante-open-source-enterprise-such-plattform/> (letzter Zugriff: 10.12.2014)

<sup>11</sup> <http://www.zweiblog.com/2012/07/solr-die-hochperformante-open-source-enterprise-such-plattform/> (letzter Zugriff: 10.12.2014)

vorhanden sein. Wenn bei der Indizierung auch als Volltext abspeichert, würde der Index sehr groß werden, was folglich Auswirkungen auf die Geschwindigkeit und Performance der Suche hat. Folglich sollte der Index möglichst übersichtlich und klein gehalten werden.<sup>12</sup> <sup>13</sup>

### 3.1.5.2 SOLR-FELDTYPEN

Hier wird Solr mitgeteilt, welche Art von Daten sich in einem Feld befindet und wie diese verarbeitet werden soll. Ein solcher fieldType Bereich kann sich dabei in verschiedenen Analyzer unterteilen. Dadurch kann innerhalb eines Feldes eine unterschiedliche Verarbeitung der Daten beim Data Import bzw. bei der Abfrage (query) erreicht werden.

In der schema.xml findet man eine Menge von vordefinierten Feldtypen. Es gibt eine Reihe von Feldern, wie int oder date, die auf Standardtypen basieren. Es gibt aber auch viele Felder, wie beispielsweise text\_de oder text\_general, die auf dem Feldtyp solr.TextField aufbauen. Auf den Fieldtype text\_general wird im Abschnitt 3.1.5.5 näher eingegangen.

Die Feldtypen in der schema.xml werden anhand des XML-<types> definiert:

```
<fieldType name="string" class="solr.StrField" />
```

Ein Feld ist folgendermaßen aufgebaut:

- *name*: Ein eindeutiger Name des Feldtypes, welcher bei der Definition eines Feldes genutzt wird.
- *class*: Der Name der Java-Implementierungsklasse des Feldtypes

Hier können auch Eigenschaften (Properties) definiert werden, die auch einen Einfluss auf die Auffindbarkeit haben. Man kann diese Properties in den Felddefinitionen wieder überschreiben.

#### Beispiele für Properties:

- *SortMissingLast*:  
→ hat einen Einfluss auf die Sortierung von Dokumenten. Wird diese Property auf true gesetzt ist, werden Dokumente ohne dieses Feld an das Ende der Sortierung platziert  
Beispiel von Felddefinition in schema.xml:

```
<fieldType name="string" class="solr.StrField" sortMissingLast="true"/>
```

- *SortMissingFirst*:  
Ähnlich wie „sortMissingLast“, nur dass die Dokumente an den Anfang gepackt werden.
- *Indexed*  
→ setzt man diese Property auf true, so wird in dem Feld gesucht

<sup>12</sup> <http://php-solr-lucene.blogspot.de/2011/06/schemaxml.html> (letzter Zugriff: 01.03.2015)

<sup>13</sup> Klose, M., & Wrigley, D. (2014). *Einführung in Apache Solr*. O'Reilly Germany

- *Stored:*  
→ setzt man diese Property auf true, so wird dieses Feld in der Trefferliste angezeigt.
- *MultiValued:*  
Dies muss man auf true setzen, damit ein Feld mehrere Werte haben kann.
- *PositionIncrementGap:*  
Hier wird bestimmt, wie viel Platz zwischen den einzelnen Werten im Feld gelassen wird. Das wirkt sich auf Feldern aus, die multiValued=true sind und in denen Phrasensuche durchgeführt wird.<sup>14</sup>

```
<fieldType name="int" class="solr.TrieIntField" precisionStep="0"
positionIncrementGap="0"/>
```

```
<fieldType name="date" class="solr.TrieDateField" precisionStep="0"
positionIncrementGap="0"/>
```

Es können mehrere Properties je nach Typ eingefügt werden.

Die Felder kann man in folgende Typen einordnen:

- String und Text
- Boolean
- Nummerische Typen
- Datum
- usw...

### 3.1.5.3 ALLGEMEINE EINSTELLUNGEN

Im nächsten Teil der Dokumentation werden die Einstellungen erläutert, die Folgen auf die Suche und die Auffindbarkeit von Dokumenten haben.

#### Der Analyse-Prozess

Die Analyse wirkt sich stark auf den Index und die Auffindbarkeit von Dokumenten aus. Hierbei werden aus dem Text die einzelnen Bestandteile extrahiert und bereitet es für die Suche vor. Hieraus entstehen drei Phasen:

- Charfilter
- Tokenizer
- Tokenfilter

In diesem Prozess wird zunächst der Text in Tokens verwandelt. Tokens bezeichnen hier die einzelnen Wörter, die durch Leerzeichen getrennt werden. Im nächsten Schritt werden diese

---

<sup>14</sup> Klose, M., & Wrigley, D. (2014). *Einführung in Apache Solr*. O'Reilly Germany

vorbereitet und mit Zusatzinformationen (z.B. Offsets, Positionsinformationen) erweitert und in den Index gelegt.

Der ganze Analyse-Prozess wird sowohl zur Indexierungszeit als auch jedes Mal zur Suchzeit durchgeführt. Dies führt zu einer Normalisierung des Texts und erhöht die Trefferwahrscheinlichkeit.

Der Vorgang wird nur für Felder durchgeführt, bei denen die Property „indexed“ auf true gesetzt wurde. Für den Fall, dass auch „stored“ auf true gesetzt ist, ist für die Anzeige der nicht analysierte Text zusätzlich vorhanden.

## Konfiguration des Analyse-Prozess

Für die Konfiguration wird innerhalb des <fieldType>-Elements ein <analyzer>-Element definiert.

### 1. Schritt: CharFilter:

Der erste Schritt bei der Analyse betrifft die Charfilter. Der Einsatz der Charfilter ist optional, d.h es können 0 bis n Charfilter in die Analyse eingebaut werden. In der Regel ist nur ein Charfilter im Einsatz. In dieser schema-Datei wurde jedoch kein Charfilter angewendet.

### 2. Schritt Tokenizer:

Der nächste Schritt der Analyse bezieht sich auf die Verarbeitung des Textes. hierbei wird der Text in einzelne Teile zerlegt, sogenannte Tokens. Dieser Vorgang ist zwingend notwendig und erstellt eine Liste von Tokens aus der generierten Zeichenkette, die als Input dient.

Im folgenden Abschnitt werden die möglichen Implementierungen für verschiedene Einsatzmöglichkeiten erläutert, die auch in der schema.xml des Projektes verwendet werden.

## Einige Beispiele:

- *WhitespacetokenizerFactory:*

Dieser Tokenizer zählt als sehr einfach und ist für die Teilung des Texts an den Leerzeichen zuständig und entfernt diese. Jedoch beachtet dieser Tokenizer die Zeichensetzung (Kommata, Semikolon etc.) nicht und deshalb werden auch weitere Filtere eingesetzt

- *KeywordTokenizerfactory:*

Mit diesem Tokenizer wird dafür gesorgt, dass aus dem gesamten Input genau ein Token entsteht und der Inhalt des Inputs genau so übernommen wird wie er ist.

- *StandardTokenizerfactory:*

Ein Tokenizer, der bei Sprachen eingesetzt wird, bei denen die Wörter durch Leerzeichen getrennt sind. Außerdem entfernt er Punktierungen, wenn nach dem Zeichen ein Leerzeichen kommt.

- *ClassicTokenizerfactory:*

Dieser Tokenizer funktioniert ähnlich wie der StandardTokenized. Es zerlegt das text

field in Tokens, indem es Leerzeichen und Punkturierung als Abtrennung behandelt. Die Abtrennungen erfolgt nach folgenden Kriterien:

- Punkte, die nicht nach einem Leerzeichen folgen
- Wörter, die durch Bindestriche getrennt sind
- erkennt Internetadressen und Email-Adressen als einen einzigen Token<sup>15</sup>

Für weitere FilterFactories siehe:

<https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters#solr.ClassicTokenizerFactory>

### *3. Schritt Tokenfilter*

In diesem Schritt können 0 bis n Tokenfilter definiert und alle durchlaufen werden. Hierbei erhält der Filter als Input eine Liste von Tokens, die sie beeinflussen. Nach der Durchführung sind Terme übrig bzw. Textbausteine, die im Nachhinein im Index abgelegt werden. Tokens können durch den Tokenfilter gelöscht, verändert oder neu generiert werden.

Die Reihenfolge bei der Definition ist von großer Bedeutung, da TokenFilter nacheinander ausgeführt werden müssen.

Einige Beispiele, auf die im späteren Verlauf näher eingegangen werden wird:

- Lowercasefilterfactory
- ClassicFilterFactory
- StopFilterfactory<sup>16</sup>

#### **3.1.5.4 QUERY UND INDEX**

In Solr ist es möglich die zu verwendeten FilterFactories in den Query- (zum Abfragen der Daten) oder den Indexteil (für das Indizieren der Daten) zu schreiben. Es ist auch möglich FilterFactories in beide Teile zu schreiben, jedoch kann dies zu einer schlechteren Performance führen.

Obwohl der Analyseprozess sowohl für das Indexieren (indexing), als auch für Anfragen (querying) verwendet wird, müssen nicht dieselben Analyseeigenschaften für beide Operationen verwendet werden. Für das Indexieren möchte man Wörter oft vereinfachen oder normalisieren, wie zum Beispiel alle Buchstaben in die Kleinschreibung umformen, Satzzeichen und Akzente zu beseitigen oder Wörter auf ihren Stamm zu reduzieren. Auf diese Weise kann die Anzahl der passenden Ergebnisse erhöht werden, weil somit zum Beispiel die Wörter "ram", "Ram" und "RAM" alle gleich behandelt werden würden. Um die Anfragezeitgenauigkeit (query) zu erhöhen, kann ein Filter eingesetzt werden um zum Beispiel alle Abkürzungen zu ignorieren.

---

<sup>15</sup> <https://archive.apache.org/dist/lucene/solr/ref-guide/apache-solr-ref-guide-4.7.pdf> (letzter Zugriff: 05.03.2015)

<sup>16</sup> Klose, M., & Wrigley, D. (2014). *Einführung in Apache Solr*. O'Reilly Germany

### 3.1.5.5 FIELDTYPE TEXT\_GENERAL

In diesem Feldtyp werden allgemeine Einstellungen für die Sprachverarbeitung bei Solr in den Suchabfragen eingestellt. Dies ist das entscheidende Feld, in dem gearbeitet und Veränderungen vorgenommen wurden, um die Anforderungen umzusetzen und eine leistungsfähige Suchmaschine zu entwickeln.

In der fertigen Version gibt es 2 analyzer. Einer vom type="index" (für das Indexieren der Daten), der andere vom type="query" zum Abfragen der Daten. Dies ist die Aufteilung der Filter in den Query- oder Indexteil. Der *tokenizer* ist dabei das Element, mit dem Solr / Lucene Textblöcke in einzelne Worte spaltet.

Danach folgen Filter.

#### 3.1.5.5.1 INDEX

Beim index analyzer werden also die zu importierenden Texte erst mittels *tokenizer* in einzelne Worte zerlegt.<sup>17</sup>

##### ClassicTokenizerfactory

Der ClassicTokenizer hat eine ähnliche Aufgabe wie standardTokenizer, und zwar zerlegt dieser Zeichen nach einem sinnvollen Muster. Einige TokenTypes sind number, alphanumeric, email, acronym, URL, etc. —

Beispiel: "I.B.M. cat's can't" => ACRONYM: "I.B.M.", APOSTROPHE:"cat's", APOSTROPHE:"can't"

```
<tokenizer class="solr.ClassicTokenizerFactory"/>
```

Darauf folgen die verschiedenen Filter.

##### Lower Case Filter Factory

Nach einigen durchgeführten Tests mit den ExampleDocs, wurde folgendes festgestellt. Die aktuelle Grundkonfiguration nimmt bei Suchanfragen keine Rücksicht darauf, ob man das zu suchende Wort klein oder groß schreibt (Caseing).

Dies wird durch die LowerCaseFilter Factory geregelt.

Bsp.: es ist egal, ob man nach Ipod, iPod, ipod oder ähnlichem sucht. Das Suchergebnis bleibt das gleiche.

Für die Anforderungen ist dies wünschenswert, deswegen wurde diese Einstellung beibehalten.

```
<filter class="solr.LowerCaseFilterFactory"/>
```

<sup>17</sup> Beispielhafte Anpassung der Schema.xml

## Synonym Filter Factory (für die Synonymerkennung zuständig)

Wie der Name vermuten lässt, indiziert dieser Filter Synonyme für ein gegebenes Wort. Besonders wichtig dabei ist die Datei, welche die Synonyme enthält, diese muss zusammen mit dem Filter eingebunden werden. Jede Zeile sollte dabei eine Synonymkette enthalten, z.B. „Fortführung, Wiederaufnahme“. Das führt dazu, dass eine Suche nach Fortführung auch Texte findet, die Wiederaufnahme enthalten.

Verwendung des Filters:

```
<filter class="solr.SynonymFilterFactory"
synonyms="synonymsGerman.txt" ignoreCase="true" expand="true"/>
```

Auszug aus der Synonymtextdatei (synonymsGerman.txt, 74 KB):

<i>Akteur, Schauspieler</i>
<i>tatsächlich, wirklich</i>
<i>Inserat, Reklame, Werbung, Zeitungsanzeige</i>
<i>addieren, zurechnen</i>
<i>Ergänzung, Zugabe</i>
<i>erfahren, geschickt</i>
<i>zugeben, zulassen, anerkennen</i>
<i>Gewinn, Vorteil</i>
<i>zustimmen, übereinstimmen</i>
<i>Abkommen, Vertrag</i>
<i>Hilfe, Mithilfe, Unterstützung</i>
<i>Flughafen, Flugplatz</i>
<i>lebend, lebendig</i>
<i>stets, immer</i>
<i>Dichtband, Dichtungsband</i>
<i>Mensch der Altsteinzeit, Urmensch</i>
<i>Ankerplatz, Ankergrund</i>
<i>Adressat, Mandant, Auftraggeber, Kunde</i>

## Stop Filter Factory

Es gibt viele Wörter, die bei einer Suche keine Beachtung finden sollten. Das liegt daran, dass jedes Suchergebnis von Solr mit einer Relevanz belegt wird. Diese resultiert u.a. auch daraus, wie oft ein Wort im indizierten Text vorkommt. Sucht also ein Nutzer nach „A und B“, wird per Default eine Oder-verknüpfte Suche durchgeführt. Es wird also nach „A“, „und“ und „B“ gesucht. Vermutlich werden viele Suchergebnisse das Wort „und“ enthalten, aber die für den Nutzer eigentlich relevanteren Inhalte „A“ und „B“ tauchen gar nicht oder nur selten in der Ergebnisliste auf. Um diese überwiegend unbrauchbaren Suchergebnisse gar nicht erst eintreten zu lassen, wird die Stop Filter Factory verwendet. Diese bezieht sich auf eine Datei, die eine erweiterbare Liste von Wörtern enthält, die von der Suche ausgeklammert werden sollen. Somit kann die Qualität der Suchergebnisse durch die Qualität Liste an Stoppwörtern beeinflusst werden.

Verwendung des Filters:

```
<filter class="solr.StopFilterFactory" words="stopwordsGerman.txt"
ignoreCase="true" enablePositionIncrements="true"/>
```

Auszug aus der Stopfiltertextdatei (stopwordsGerman.txt, 2,37 KB):

```
anderes
anderem
andern
anderer
anders
auch
auf
aus
bei
bin
bis
bist
da
```

## German Normalization Filter Factory

Dieser Filter berücksichtigt Besonderheiten der deutschen Sprache, so ist dieser Filter auch für Umlaute zuständig.

Umlaute werden von Solr in der Beispielkonfiguration nicht erkannt. Bei Tests kam heraus, dass „ü“ als ue gelesen wird, sucht man nach „Uebung“ kommt sowohl das Dokument mit dem Inhalt „Uebung“, als auch das Dokument mit dem Inhalt „Übung“. Dies ist soweit Wünschenswert, falls der Suchende nicht weiß wie z.B. ein „Schröder“ geschrieben wird. Sucht man allerdings nach „Übung“ findet Solr keine Ergebnisse.

Für die Funktion von Umlauten ist die German Normalization Filter Factory von Bedeutung. Richtig eingesetzt, transformiert er Umlaute (z.B. ä wird zu ae).

```
<filter class="solr.GermanNormalizationFilterFactory"/>
```

## Snowball Porter Filter Factory

Der SnowballPorterFilter erlaubt das Stemming von Wörtern einer beliebigen Sprache, ganz abgesehen davon, ob das Wort in einem offiziellen Wörterbuch existiert oder nicht.

Obwohl der Algorithmus für jede Sprache unterschiedlich ist, wird die Snowball Stammerkennung als sehr aggressiv eingestuft, da der Filter oft große Suffixe entfernt, um Wörter zu einem Stamm zu verringern. Dies kann den positiven Effekt der Verringerung der Zahl der Begriffe im Index haben. Jedoch kann dies zu Suchergebnissen führen, die in keinem Zusammenhang mit der ursprünglichen Abfrage stehen.

Wie zum Beispiel das Entfernen von „national“ im Wort „international“. Ein weiterer Nachteil dieses Algorithmus ist die Reduktion von Begriffen, die denselben Stamm haben,

sich aber nur in Zeit und/oder Person unterscheiden, wie zum Beispiel: „laufen“, „laufe“, „lief“, „liefen“, „gelaufen“.

Hier könnte man zusätzlich den StemmerOverrideFilterFactory nach dem Stemmer in der Analysekette verwenden.

Nachteil dieser Methode ist jedoch zudem noch eine Textdatei/Wörterbuch notwendig ist, von welchem es noch keine offizielle Version gibt (hier: override.txt), die pro Zeile eine Regel beinhaltet, die folgendermaßen aussehen muss: Das zu ersetzende Wort (z.B. „lief“) bildet den Anfang einer Zeile gefolgt von einem Tabspace, gefolgt von dem Stamm, zu dem es „reduziert“ werden soll (z.B. lauf). Also in etwa folgendermaßen:

Gelaufen	→ lauf
Lief	→ lauf
Risiken	→ Risiko

Es könnte natürlich auch vorkommen, dass eine Regel „zu aggressiv“ für die Anwendung scheint. Auch für diesen Fall ist vorgesorgt. Und zwar kann man zusätzlich den KeyWordMarkerFactory einbinden, der alle in der Datei (hier: „protwords.txt“) befindlichen und bei der Indexierung bzw. Suche vorkommenden Wörter markiert, welche somit vom Stemmer nicht berührt werden.

Filter werden wie folgt eingebunden:

```
<filter class="solr.SnowballPorterFilterFactory"  
language="German2" protected="protwords.txt"/>  
  
<filter class="solr.KeywordMarkerFilterFactory"  
protected="protwords.txt"/>  
  
<filter class="solr.StemmerOverrideFilterFactory" dictionary=  
"override.txt" ignoreCase="false"/>
```

Der solr.SnowballPorterFilterFactory stellt zwei unterschiedliche Sprachattribute bereit: „Deutsch“ und „Deutsch 2“. „Deutsch 2“ ist nur eine modifizierte Version von „Deutsch“, die die Umlaute anders behandelt: zum Beispiel zur Behandlung von „ü“ als „ue“.

Weiterer nützlicher Filter für das Stemming:

### HunspellstemFilter

Im Gegensatz zum Snowballstemmer, ist Hunspell kein rein algorithmisches System. Stattdessen verwendet er eine Datei, die die grammatischen Regeln für eine Sprache in einem standardisierten Format codiert, und eine für die Sprache gültige Wörterbuch-Datei. Bei der Analyse eines Wortes, geht Hunspell über die grammatischen Regeln und prüft dann, ob ein gültiger Stamm gefunden wird. Der Vorteil dieses Verfahrens ist, dass sehr komplizierte

grammatische Regeln angewendet werden können wie die Entfernung mehrerer Suffixe und Präfixe und das Hinzufügen von neuen Zeichen. Außerdem wird Hunspell solange suchen, bis mehrere gültige Wortstämme gefunden werden, statt nur einem. Doch im Gegensatz zu Snowball, der jedes beliebige Wort analysieren kann, findet Hunspell nur Wörter, die seinen Regeln entsprechen oder im Wörterbuch existieren.

Es existieren bereits hunderte Hunspell-Wörterbücher für andere Sprachen als Englisch, die jedoch leider in der Qualität erhebliche Unterschiede aufweisen und noch nicht ganz ausgereift sind. Ebenso gibt es einige Wörterbuchformate, die noch nicht unterstützt werden. Somit variiert die Hunspell Eignung in Abhängigkeit von der Qualität der Wörterbücher.

Der Filter wird wie folgt eingebunden:

```
<filter class="solr.HunspellStemFilterFactory"  
dictionary="en_GB.dic" affix="en_GB.aff" ignoreCase="true"/>
```

### **Remove Duplicates Token Filter**

Filtert alle Tokens heraus, die in der gleichen logischen Position im tokenstream sind, wie frühere Tokens mit dem gleichen Text.

```
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
```

#### 3.1.5.5.2 QUERY

Beim Query-Analyzer werden zunächst die zu importierenden Texte mittels Tokenizer in einzelne Worte zerlegt.<sup>18</sup>

### **ClassicTokenizerfactory (siehe Informationen im Index)**

```
<tokenizer class="solr.ClassicTokenizerFactory"/>
```

Darauf folgen die verschiedenen Filter.

### **Word Delimiter Filter Factory**

Dieser Filter ist für die Worttrennung bei zusammengesetzten Wörtern zuständig. Durch diesen Filter wird nicht nur gesamte Wort gesucht, sondern auch Webseiten und Dokumente mit Teilwörtern des Suchwortes angezeigt. Folgender Quellcode-Ausschnitt zeigt, wie eine solche Einbindung von einem TokenFilter aussehen kann:

```
<filter class="solr.WordDelimiterFilterFactory"/>
```

---

<sup>18</sup> Beispielhafte Anpassung der Schema.xml

Parameter/Attribute:

`generateWordParts="1"` generiert einzelne Wortteile:

- "PowerShot" => "Power" "Shot" (falls `splitOnCaseChange=1`)
- "Power-Shot" => "Power" "Shot"
- Voreinstellung ist 1

`generateNumberParts="1"` generiert einzelne Ziffernblöcke:

- "500-42" => "500" "42"
- Voreinstellung ist 1

`catenateWords="1"` regelt die maximale Anzahl der zu verkettenden Wörter:

- "wi-fi" => "wifi"
- Voreinstellung ist 0

`catenateNumbers="1"` regelt die maximale Anzahl der zu verkettenden Zahlen:

- "500-42" => "50042"
- Voreinstellung ist 0

`catenateAll="0"` verkettet alle Teilwörter:

- "wi-fi-4000" => "wifi4000"
- Voreinstellung ist 0

`splitOnCaseChange="1"` verursacht Kleinschreibung => Großschreibung um neue Teile zu generieren:

- "PowerShot" => "Power" "Shot"
- "TransAM" => "Trans" "AM"
- Voreinstellung ist ("1"); stelle auf 0 um auszuschalten

`preserveOriginal="1"` erlaubt das Indexieren von originalen Tokens ohne Modifizierung (zusätzlich zu den Tokens, welche bereits durch andere Optionen erzeugt wurden) Voreinstellung ist 0

`protected="protwords.txt"` spezifiziert eine Textdatei, die eine Liste von Wörtern enthält, die geschützt und unverändert übergeben werden sollen.

- Voreinstellung ist leer (keine geschützten Wörter)

`types="wdfftypes.txt"` ermöglicht spezifische tokenizer für diesen Filter. Diese Datei sollte im solr/conf Ordner gespeichert werden und Einträge müssen in folgender Form sein (ohne Anführungszeichen):

"%" => ALPHA" or "\u002C => DIGIT". Allowable types are: LOWER, UPPER, ALPHA, DIGIT, ALPHANUM, SUBWORD\_DELIM. [Solr3.1]

Getrennt wird mit den folgenden Regeln:

- separieren an Trennsymbolen innerhalb des Wortes (alle nicht alphanumerisch Zeichen).
  - "Wi-Fi" -> "Wi", "Fi"
- Trennung bei Wechsel von Klein- zu Großschreibung
  - "PowerShot" -> "Power", "Shot"
  - kann mit splitOnCaseChange eingestellt werden
    - splitOnCaseChange="1" aktiviert
    - splitOnCaseChange="0" deaktiviert
- Trennen bei Wechsel von Zahlen und Buchstaben
  - "SD500" -> "SD", "500"
  - kann mit splitOnNumerics eingestellt werden
    - splitOnNumerics="1" aktiviert
    - splitOnNumerics="0" deaktiviert
- Zeichen vor und nach Wörter werden nicht beachtet
  - "//hello---there, 'dude'" -> "hello", "there", "dude"
- nachfolgende "s" werden nicht beachtet
  - "O'Neil's" -> "O", "Neil"
  - kann mit stemEnglishPossessive eingestellt werden
    - stemEnglishPossessive="1" aktiviert
    - stemEnglishPossessive="0" deaktiviert

### Lower Case Filter Factory

`<filter class="solrLowerCaseFilterFactory" />`

Zur Beschreibung siehe gleichnamigen Abschnitt unter 3.1.5.5.1 Index.

### Dictionary Compound Word Token Filter Factory

Dieser Filter hilft dabei, zusammengesetzte Wörter in ihre Teilwörter zu trennen. So können aus "Donau-Dampfschifffahrt" die Teilwörter "Donau", "Dampf", "Schiff" und "Fahrt" extrahiert werden. Voraussetzung hierfür: Die Wörter sind im eingebundenen Wörterbuch enthalten. In den Suchergebnissen werden so auch Texte aufgeführt, die Donau-Dampfschifffahrt enthalten, auch wenn nur nach einem Teil des Wortes, z.B. „Schiff“, gesucht wurde. Um unscharfe Suchergebnisse zu vermeiden, sollte dieser Filter aber nur zur Indizierungszeit angewendet werden. Um es an einem Beispiel zu erklären, wird eine

Auflistung von Treffern zur "Fahrt" vermieden, wenn nach "Donau-Dampfschiffahrt" gesucht wird. Damit das gut funktioniert, muss das benutzte Wörterbuch möglichst umfangreich sein.

Verwendung des Filters:

```
<filter class="solr.DictionaryCompoundWordTokenFilterFactory"
       dictionary="dictionaryGerman.txt"
       minWordSize="5"
       minSubwordSize="3"
       maxSubwordSize="30"
       onlyLongestMatch="false"/>
```

#### Parameter:

- `input` - Der zu verarbeitende Eingabefluss (hier Tokenstream)
- `dictionary` - Das Wörterbuch mit welchem verglichen wird
- `minWordSize` - Nur Wörter, die länger sind als das werden bearbeitet
- `minSubwordSize` - Nur Teilwörter, die länger sind als das, kommen in den Ausgabestrom ("Outputstream")
- `maxSubwordSize` - Nur Teilwörter, die kürzer sind als das, kommen in den Ausgabestrom ("Outputstream")
- `onlyLongestMatch` - Fügt nur die längsten passenden Teilwörter zum Ausgabestrom ("Outputstream")

#### German Normalization Filter Factory (siehe Definition im Index)

Dieser Filter berücksichtigt Besonderheiten der deutschen Sprache. Richtig eingesetzt, transformiert er Umlaute (z.B. ä wird zu ae).

```
<filter class="solr.GermanNormalizationFilterFactory" />
```

#### Snowball Porter Filter Factory

```
<filter class="solr.SnowballPorterFilterFactory" language="German2"
protected="protwords.txt"/>
```

Zur Beschreibung siehe gleichnamigen Abschnitt unter 3.1.5.5.1 Index.

#### Remove Duplicates Token Filter (siehe Definition im Index)

```
<filter class="solr.RemoveDuplicatesTokenFilterFactory" />
```

weitere Anpassungsmöglichkeiten gibt es im Solr.Wiki<sup>19</sup>

---

<sup>19</sup><https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters#solr.ClassicFilterFactory> (letzter Zugriff: 23.01.15)

### 3.1.6 KONFIGURATIONSDATEI

Die „solrconfig.xml“ Datei enthält Parameter und Einstellungen mit denen Solr selbst konfiguriert werden kann. Die solrconfig.xml befindet sich bei diesem Projekt im Ordner: „SolrStack/solr/collection1/config“.

Sie beinhaltet alle wichtigen Methoden für die meisten Installationen. Jeder Solr-Core hat seine eigene Konfigurationsdatei. Bei allen Konfigurationen in der solrconfig.xml wird ".solr" statt "org.apache.solr." benutzt. Spezifiziert man seine eigenen Plugins muss das beachtet werden und gegebenenfalls muss die eigene Java-Klasse vollständig angegeben werden.

Für diesen Teil der Dokumentation wurden wiederkehrend die gleichen Quellen genutzt.<sup>20 21</sup>  
<sup>22 23 24 25 26</sup>

Folgende grundlegende Einstellungen können in der solr.config angepasst werden:

- Allgemeine Einstellungen
- Index-Einstellungen
- Query-Einstellungen
- RequestHandler
- UpdateHandler

#### 3.1.6.1 ALLGEMEINE EINSTELLUNGEN

Die benutzte Apache Lucene Version von Solr kann Einfluss auf die verwendeten Funktionalitäten haben. Aktualisiert man die Lucene Version sollte man beachten, dass zumeist auch der Index aktualisiert werden muss. Die verschiedenen Versionen sind abwärts kompatibel, so können neuer Solr Instanzen mit älteren Lucene Indexen arbeiten. Lucene selbst empfiehlt immer die aktuelle Version zu benutzen, da man so von allen Verbesserungen und Fehlerbehebungen profitieren kann.

#### Bibliotheken

Bibliotheken können genutzt werden, um festzulegen von welchem Ort oder von welchem Ordner „jars“ geladen werden sollen. So können jegliche Plugins, die in der schema oder in solrconfig aufgeführt wurden, behandelt werden. Bibliotheken werden in der solrconfig.xml eingebunden, indem sie durch <lib /> definiert werden. Die Plugins werden in der Reihenfolge, wie sie in solrconfig.xml aufgelistet sind, geladen. Gibt es Abhängigkeiten in der Liste, wird der niedrigste Level zuerst geladen.

<sup>21</sup> <http://wiki.apache.org/solr/SolrConfigXml>; (letzter Zugriff: 06.03.2015)

<sup>22</sup> Apache Solr Ref Guide 4.10

<sup>23</sup> <https://wiki.apache.org/solr/Suggester> (letzter Zugriff: 28.02.2015)

<sup>24</sup> <http://wiki.apache.org/solr/SolrRequestHandler> (letzter Zugriff: 28.02.2015)

<sup>25</sup> <http://www.dmk-ebusiness.de/blog/artikel/was-hat-solr-vorzuschlagen/> (letzter Zugriff: 03.03.2015)

<sup>26</sup> [http://lucene.apache.org/solr/4\\_10\\_1/tutorial.html](http://lucene.apache.org/solr/4_10_1/tutorial.html) (letzter Zugriff: 03.03.2015)

```
<lib dir="../../../../contrib/extraction/lib" regex=".*\.\.jar" />
<lib dir="../../../../dist/" regex="solr-cell-\d.*\.\.jar" />
```

Die Bibliotheken können absolut oder relativ angegeben werden. Existiert ein „./lib“ Verzeichnis in der „instanceDir“, werden alle dort gefundenen Dateien eingebettet, dies drückt also das gleiche aus wie „<lib dir = ./lib />“ in der solrconfig.xml. Die Angabe von „dir“ ermöglicht, dass jede Datei aus dem zum Pfad gehörenden Verzeichnis geladen wird. Dies ist eine einfache Lösung alle „jars“ eines Verzeichnisses einzubinden. Wird zusätzlich zum „dir“ auch noch „regex“ festgelegt, werden nur die Dateien die genau dem „regex“ entsprechen geladen. „regex“ gibt also eine Namenskonvention an. Wird „dir“ angegeben und keine passenden Dateien gefunden, wird eine Fehlermeldung ausgegeben.

## Data Dir

Mit dem „DataDir“ Parameter kann man den Ort der Speicherung von Indexdaten angeben. Standardmäßig speichert Solr die Indexdaten in einem Verzeichnis namens „/data“ im solrHome Ordner. Es ist möglich ein anderes Verzeichnis zum Speichern der Indexdaten zu verwenden. Ein anderes Verzeichnis kann entweder mit einem vollständigen Pfadnamen oder einen Pfad relativ zum aktuellen Arbeitsverzeichnis angegeben werden. In diesem Projekt wurde sich für die interne Variante entschieden, da eine möglichst integrierte Lösung angeboten werden soll.

```
<dataDir>${solr.data.dir:}</dataDir>
```

## Directory Factory

Möchte man die Indexdaten nicht intern in der Solr Konfiguration speichern, kann man dafür die Directory Factory benutzen. Die Standard „solr.StandardDirectoryFactory“ basiert auf einem Dateisystem und versucht die bestmögliche Implementierung für die aktuelle JVM und die Plattform auszusuchen. Es kann eine bestimmte Implementierung erzwungen werden, indem man „solr.MMapDirectoryFact“, „solr.NIOFSDirectoryFactory“ oder „solr.SimpleFSDirectoryFactory“ angibt.

```
<directoryFactory name="DirectoryFactory"
class="${solr.directoryFactory:solr.NRTCachingDirectoryFactory}">
```

## Inverted Index

Der Index von Solr wird als invertiert bezeichnet, da nicht Seiten die auf Wörter zeigen gespeichert werden, sondern Wörter die auf die zugehörige Seite verweisen. Die "CodecFactory" definiert das Format des Invertierten Indexes. Als Standard ist die "SchemaCodecFactory" eingestellt.

## Managed Schema Definition

Die Schema-API ermöglicht Schemaänderungen über eine REST-Schnittstelle. Es gibt auch die Möglichkeit einen Nur-Lese-Zugriff auf alle Schema Elemente zu erlauben. Es gibt Schwierigkeiten damit, eine zugriffsgesteuerten Kontrolle bei einer Konfigurationsdatei zu ermöglichen, wenn diese auch offen für manuelle Änderungen ist.

System generierte und manuelle Änderungen können sich überschneiden und die vom System generierten Bearbeitungen könnten Kommentare oder andere Anpassungen entfernen. Diese können entscheidend sein, um zu verstehen welche Felder, Feldtypen, etc. es gibt und warum sie auf diese Art und Weise festgelegt wurden. Damit dieses Problem nicht auftreten kann, müsste eine Versionierung eingeführt werden. Solrconfig.xml erlaubt die Solr-Schema als "Managed Schema" zu definieren. So sind Schemaänderungen nur durch die Schema-API möglich. ClassicIndexSchemaFactory ermöglicht die manuelle Änderung der schema.xml. Da der fertige Suchservice ein voll funktionsfähiger Service ist, soll die schema.xml nicht umgehend weiter konfiguriert werden müssen. Der Fall das zum gleichen Zeitpunkt mehrere Personen den gleichen Abschnitt bearbeiten, ist somit eher unwahrscheinlich, sodass Schemaänderungen weiterhin manuell zugelassen werden.

### 3.1.6.2 INDEX

Mit diesen Einstellungen kann das Indizieren der Dateien und der Aufbau des Indexes beeinflusst werden. Der Abschnitt der Indexverwaltung in der solrconfig.xml wird mit <indexConfig> eingeleitet. In der Beispielkonfiguration von Solr werden diese Einstellungen in der Regel auskommentiert, das bedeutet, dass die Standardeinstellung verwendet wird. Für den Normalgebrauch sind diese Einstellung vollkommen ausreichend.

Anwendungsgebiete von IndexConfig:

- Sizing von Index-Abschnitten
- Zusammenführen von Index-Abschnitten
- Index Locks
- Andere Index-Einstellungen

Die wichtigsten Parameter um den Index zu beeinflussen werden folgend beschrieben. Obwohl alle die Suche optimieren und an den eigenen Bedarf anpassen können, wurden nicht alle möglichen Einstellungen in der fertigen Konfiguration implementiert.

#### LimitTokenCount

Dieser Parameter gibt an wie viele Tokens maximal je Feld gespeichert werden dürfen. Dadurch ist es möglich die Größe des Indexes zu verringern. Tokens die die maximal definierte Anzahl überschreiten, werden nicht im Index gespeichert.

#### writeLockTimeOut

Es gibt eine Sperre, die verhindert, dass mehr als eine Person gleichzeitig den Index verändern. Die Sperre wird aktiv sobald eine Person anfängt Dokumente zum Index hinzuzufügen oder zu löschen. Mit diesem Parameter wird die Zeit festgelegt, wie lange auf eine solche Sperre gewartet wird.

## **maxIndexingThreads**

Mit diesem Parameter kann man die Schnelligkeit der Indizierung steuern. Es wird die Anzahl der gleichzeitig laufenden Threads bei der Indizierung festgelegt. Ist der Grenzwert erreicht, werden die zusätzlichen Threads warten bis die anderen beendet wurden.

## **useCompoundFiles**

Sollen mehrere Dateien eines Abschnittes in einer einzelnen Datei, statt in mehreren gespeichert werden, setzt man diesen Parameter auf „true“. Damit ist es möglich die Anzahl der Dateien im Index zu verringern, die Laufzeit der Suche ist dadurch möglicherweise länger. Von Lucene selbst wird dies als eine Expertenanpassung bezeichnet. Dieser Parameter ist nicht aktiviert, da die Laufzeit der Suche minimiert werden soll.

## **RamBufferSizeMB/maxBufferDocs**

RamBufferSizeMB definiert die Größe des RAMs, die zum Indizieren von gebuffernten Dokumenten, die hinzugefügt oder gelöscht werden, genutzt wird, bevor sie ins Verzeichnis gespeichert werden. MaxbufferDocs definiert die maximale Anzahl der gebuffernten Dateien bevor gespeichert wird. Sobald eines der beiden Limits erreicht wird, wird gesichert. Der Standardwert ist 100MB. Je höher die Werte, desto schneller geht die Indizierung. Nach einer Sicherung sind die Dokumente nicht durchsuchbar, erst nach einem Commit bzw. Optimize. Beim Suchservice gibt es die Möglichkeit Inhalte per Hand zu indizieren, damit diese sofort zu finden sind, ist weiterhin der Standardwert eingestellt.

## **MergePolicy**

Mit diesen Regeln wird kontrolliert inwiefern das Zusammenfügen von Abschnitten durchgeführt wird. Wird ein Update am Index durchgeführt, wird dies zum zuletzt geöffneten Segment hinzugefügt. Der Standardwert in Solr ist die „TieredMergePolicy“. Die gleich große Segment miteinander verbindet. Um die Merge-Regeln weiter zu definieren gibt es zum Beispiel den MergeFactor, dieser gibt an, wie viele Segmente Lucene haben darf bevor sie zu einem zusammengefügt werden, der Standardwert hierfür sind 10 Dateien. Das Zusammenfügen von Segmenten dauert eine gewisse Zeit, verlängert die Indizierungszeit und kann mit diesen Einstellung angepasst werden.

## **LockFactory**

Bei der Änderung am Index muss der Indexwriter eine exklusive Sperre für das Verzeichnis erlangen. Die implementierte LockFactory gibt an welche der möglichen Factories eingebunden wird, um dies zu ermöglichen. Standardmäßig ist dies die "native" Implementierung. Dies ist auch in der fertigen Konfiguration eingestellt.

```
<lockType>${solr.lock.type:native}</lockType>
```

LockType = single: verwendet „SingleInstanceLockFactory“ und ist für einen Nur-Lese-Index oder, wenn es keine andere Möglichkeit gibt, versucht es den Index zu ändern.

LockType = native: verwendet „NativeFSLockFactory“ um eine OS Dateisperre festlegen. Sollte nicht benutzt werden wenn mehrere Solr WebAnwendungen in der gleichen JVM versuchen, einen einzelnen Index zu teilen.

LockType = simple: verwendet „SimpleFSLockFactory“, legt eine einfache Datei zur Verriegelung an.

## InfoStream

Ist dieser Parameter mit "true" aktiviert, wie es beim Suchservice eingestellt ist, werden alle relevanten Informationen während des Indizierens durch den IndexWriter im solrLog ausgegeben. Dies ist gerade wenn Probleme auftreten sehr hilfreich.

### 3.1.6.3 UPDATE HANDLER

Diese Einstellungen können die Leistung der Indexaktualisierungen beeinflussen und entscheiden wie interne Updates umgesetzt werden. Die eingesetzte Konfiguration wirkt sich nicht auf die „high level“ Einstellungen des Request Handlers aus, also auf Updates die von Benutzern gesendet werden.

## Commits

Daten die zu Solr gesendet werden, können nicht durchsucht werden, bis sie in den Index committed wurden. Der Grund dafür ist, dass in einigen Fällen Commits sehr langsam sein können. Damit Überschreibungen von potentiell wichtigen Informationen vermieden werden kann, sollten sie getrennt von anderen möglichen Commitanforderungen durchgeführt werden. Also, es ist besser, wenn die Kontrolle über die Daten geregelt wird. Zur Kontrolle der Commits-Zeiten gibt es mehrere Möglichkeiten: commit and softCommit, autoCommit, commitWithin und maxPendingDeletes.

"commit" wird meistens mit einem Benutzerbefehl mitgesandt und es wird sofort nach Laden der Daten zu Solr committed. Auch bei "soft commit" werden die Dateien sofort gespeichert, es wird aber nicht garantiert, dass die Dateien dauerhaft gespeichert sind. Wird "auto commit" benutzt, wird festgelegt, wie oft ausstehenden Updates automatisch in den Index geladen werden. Eine Alternative zu "autoCommit" ist "commitWithin", welches entweder bei einer Updateanfrage an Solr oder in einem Update RequestHandler benutzt werden kann. Der Wert von "maxPendingDeletes" gibt eine Obergrenze für die Anzahl der Löschungen von Dokumenten an die Solr puffert. Der Speicherplatz während der Indizierung wird davon beeinflusst.

## Event Listener

Im UpdateHandler Abschnitt ist es zudem möglich Update bezogene Event-Listener zu konfigurieren. Dies kann beispielsweise ausgelöst werden, nachdem ein Commit eintritt.

Der Listener wird mit der RunExecutableListener aufgerufen, die eine extern ausführbare Datei ausführt, die eine definierte Anzahl von Anweisungen hat.

### 3.1.6.4 QUERY-EINSTELLUNGEN

Regulieren jegliche Einstellungen bezüglich des Query-Teils. Die Einstellungen<sup>27</sup> beeinflussen die Art und Weise, wie Solr Abfragen verarbeitet und darauf reagiert. Diese Einstellungen werden in den untergeordneten Elementen von <query> in der solrconfig.xml konfiguriert.

#### Caches

Solr Caches sind mit einer bestimmten Instanz des Index Searcher verbunden, diese bestimmte Ansicht ändert sich während der Laufzeit des Suchers nicht. Solange der Index-Sucher verwendet wird, sind alle Elemente im Cache gültig und für die Wiederverwendung verfügbar. Caching in Solr unterscheidet sich von Caching in vielen anderen Anwendungen, die zwischengespeicherten Elemente laufen nicht nach einem Zeitintervall ab, sondern bleiben für die gesamte Lebensdauer des Index-Sucher erhalten.

Wenn ein neuer Searcher gestartet wird, behandelt der aktuelle Sucher weiter Suchdienst-Anfragen während der neue Searcher seinen Cache lädt. Wenn der neue Searcher bereit ist, wird er automatisch als neuer Searcher registriert und ersetzt er den alten Searcher. Der alte Searcher wird beendet, sobald es die Bearbeitung aller seiner Anfragen abgeschlossen ist.

In Solr gibt es drei Cache-Implementierungen:

- solr.search.LRUCache
- solr.search.FastLRUCache
- solr.search.LFUCache.

#### LRUCache

Die Abkürzung steht für LRU Least Recently Used. Wenn ein LRU-Cache voll ist, wird der Eintrag mit dem ältesten Zeitstempel letzter Zugriff vertrieben, um Platz für den neuen Eintrag zu machen. Der Nettoeffekt ist, dass Einträge, auf die häufig zugegriffen wird, tendenziell im Cache bleiben, während diejenigen, auf die nicht häufig zugegriffen wird, dazu neigen, verworfen zu werden und erst wieder aus dem Index abgerufen zu werden aus dem Index, wenn sie wieder benötigt werden.

#### FastLRUCache

Die FastLRUCache, die in Solr 1.4 eingeführt wurde, wurde entwickelt, um Lock-frei sein, sodass es für Caches, die mehrmals in einer Anfrage betroffen sind, gut geeignet ist.

---

<sup>27</sup> <https://cwiki.apache.org/confluence/display/solr/Query+Settings+in+SolrConfig> (letzter Zugriff: 19.02.15)

## **LFUCache**

Die LFUCache bezieht sich auf die am wenigsten benutzte Cache. Dies funktioniert in einer Weise, ähnlich zu der LRU-Cache, mit der Ausnahme, dass wenn der Cache voll ist, die Eingabe vertrieben wird, die am wenigsten verwendet wurde.

Die Statistik in der Solr Admin-Benutzeroberfläche zeigt Informationen über die Leistung aller aktiven Caches an. Diese Informationen können helfen, die Größen der verschiedenen Caches angemessen an die eigene Anwendung anzupassen. Wenn ein Sucher beendet wird, wird eine Zusammenfassung der Cache-Nutzung gespeichert.

Jeder Cache hat Einstellungen, um seine Ausgangsgröße (initialSize), die Maximalgröße (size) und die Anzahl der während des Aufwärmens eingesetzten Elemente (autowarmCount), zu definieren. Die LRU- und FastLRU-Cache Implementierungen können, statt eines absoluten Wertes, einen Prozentsatz für autowarmCount annehmen.

Details zu den verschiedenen Caches werden unten beschrieben:

### **filterCache**

Solr verwendet den filterCache, um Ergebnisse von Anfragen zu cachen, die die fq Suchparameter verwenden. Nachfolgende Anfragen mit den gleichen Parametereinstellungen führen zu Cache-Hits und somit zu einer schnelleren Ergebnisausgabe.

```
<filterCache class="solr.LRUCache" size="512" initialSize="512"  
autowarmCount="128" />
```

### **queryResultCache**

Dieser Cache enthält die Ergebnisse von vorherigen Suchen: geordnete Listen von Dokument-IDs ( DocList ), basierend auf einer Anfrage, einer Sortierung, und dem Umfang der angeforderten Dokumente.

```
<queryResultCache class="solr.LRUCache" size="512" initialSize="512"  
autowarmCount="128" />
```

### **documentCache**

Dieser Cache beinhaltet Lucene Document-Objekte (die gespeicherten Felder für jedes Dokument). Da Lucene's interne Dokument-IDs vergänglich sind, wird dieser Cache nicht automatisch erwärmt. Die Größe für die documentCache sollte stets größer sein, als max\_results mal die max\_concurrent\_queries, um sicherzustellen, dass Solr kein Dokument während einer Anfrage erneut abrufen muss. Je mehr Felder in einem Dokument gespeichert werden, desto höher ist die Speicherbelegung eines Caches.

```
<documentCache class="solr.LRUCache" size="512" initialSize="512"  
autowarmCount="0" />
```

## **Benutzerdefinierte Caches**

Sie können auch benannte Caches für Ihren eigenen Anwendungscode definieren. Sie können ihr verwendetes Cache-Objekt finden und verwenden, indem Sie die SolrIndexSearcher-Methoden getCache(), cacheLookup() und cacheInsert() aufrufen.

```
<cache name="myUserCache" class="solr.LRUCache" size="4096"  
initialSize="1024" autowarmCount="1024" />  
regenerator="org.mycompany.mypackage.MyRegenerator" />
```

## Query-Sizing

Die Größe der Abfragen kann reguliert werden, dazu gibt es einige Parameter in der solrconfig.xml

### maxBooleanClauses

Hier kann die maximale Anzahl von Inhalten in einer Booleschen Abfrage angepasst werden. Dies kann Auswirkungen auf Anfragen mit einer großen Anzahl von Booleschen Bedingungen haben. Wird diese Grenze überschritten, wird eine Exception ausgelöst.

### enableLazyFieldLoading

Wenn dieser Parameter auf true gesetzt ist, werden Felder, die nicht direkt angefordert werden, so langsam wie möglich geladen. Dies kann die Performance verbessern, da die meisten Anfragen nur eine kleine Teilmenge an Feldern brauchen.

### queryResultWindowSize

Wird zusammen mit dem queryResultCache genutzt, es wird eine Menge der gewünschten Anzahl von Dokument-IDs zwischengespeichert. Fordert beispielsweise eine Anfrage die Dokumente 10 bis 19, und queryWindowSize ist 50, werden die Dokumente 0 bis 49 zwischengespeichert.

### queryResultMaxDocsCached

Dieser Parameter bestimmt die maximale Anzahl von Dokumenten, die für jeden Eintrag im QueryResultCache gepuffert werden.

#### 3.1.6.5 REQUESTDISPATCHER

Der RequestDispatcher<sup>28</sup> Teil von solrconfig.xml steuert die Art und Weise wie Solr Servlets RequestDispatcher auf HTTP-Anfragen reagiert. Enthalten sind Parameter zur Definition, wie man /select URLs behandeln soll, ob es Fernstreaming unterstützt, die maximale Größe der Datei-Upserts und wie es bei Anfragen auf HTTP-Cachezeiger reagiert.

### handleSelect Element

Das erste konfigurierbare Element ist das Attribut handleSelect auf dem <RequestDispatcher> selbst. Dieses Attribut kann einen von zwei Werten einnehmen, entweder "true" oder "false". Es regelt, wie Solr auf Anfragen wie /select?qt=XXX reagiert. Der Standardwert "false" ignoriert Anfragen zu /select, falls ein RequestHandler nicht explizit mit dem Namen /select

---

<sup>28</sup> <https://cwiki.apache.org/confluence/display/solr/RequestDispatcher+in+SolrConfig> (letzter Zugriff: 24.02.15)

registriert ist. Ein "true" Wert routet Anfrageanforderungen an den mit dem qt-Wert definierten Parser.

```
<requestDispatcher handleSelect="true" >
...
</requestDispatcher>
```

### **requestParsers Element**

Steuert Werte welche in Zusammenhang mit Parsinganfragen stehen. Es ist ein leeres XML Element, welches keinen Inhalt, nur Attribute hat.

Das Attribut enableRemoteStreaming steuert das Streaming von Inhalten (erlaubt oder nicht). Auf *false* gesetzt werden Streamings nicht zugelassen. Auf *true* gesetzt (Standardwert) kann der Speicherort der Inhalte, welche gestreamt werden sollen, durch Verwendung der Parameter *stream.file* oder *stream.url* definiert werden.

Bei der Aktivierung von Remote-Streaming, sollte gleichzeitig die Authentifizierung aktiviert werden. Andernfalls könnte jemand möglicherweise, durch Zugriff auf beliebige URL's, Zugriff auf die Inhalte erhalten. Sinnvoll wäre es, Solr hinter einer Firewall zu plazieren, um es von nicht vertrauenswürdigen Clients zu schützen.

Das Attribut multipartUploadLimitInKB setzt eine Obergrenze in Kilobyte auf die Größe eines Dokuments, das in einer mehrteiligen HTTP POST-Anforderung abgegeben werden kann. Das Attribut formDataUploadLimitInKB setzt eine Grenze in Kilobyte auf die Größe von Formulardaten (*application/x-www-form-urlencoded*) welche in einer HTTP POST-Anforderung abgegeben werden, welche verwendet werden kann um Anfragenparameter zu übertragen, die nicht in eine URL passen.

Das Attribut addHttpRequestToContext kann dazu verwendet werden, anzugeben, dass das ursprüngliche HttpServletRequest Objekt in der Kontextzuordnung des SolrQueryRequests enthalten sein muss, mithilfe des Schlüssels *httpRequest*. Dieser HttpServletRequest wird von keiner Solr Komponente benutzt, könnte jedoch hilfreich bei der Kundenentwicklung von Plugins sein.

```
<requestParsers enableRemoteStreaming="true"
                 multipartUploadLimitInKB="2048000"
                 formDataUploadLimitInKB="2048"
                 addHttpRequestToContext="false"/>
```

### **httpCaching Element**

Das *<httpCaching>* Element steuert HTTP Cache Control-Header. Es steuert das Caching von HTTP Antworten, welche vom W3C HTTP Spezifikationen definiert sind.

Es erlaubt drei Attribute und ein Unterelement. Die Attribute des *<httpCaching>* kontrollieren, ob eine 304 Antwort auf eine GET-Anfrage erlaubt ist, und wenn ja, welche Art von Antwort das sein sollte. Wenn eine HTTP Client Anwendung ein GET ausgibt, kann sie optional festlegen, dass eine 304 Antwort akzeptabel ist, falls die Ressource seit dem letzten Abruf nicht modifiziert wurde.

*never304*: Auf den Wert true gesetzt, wird niemals eine GET-Anfrage mit dem 304 code antworten, auch wenn die angeforderte Ressource nicht modifiziert wurde. Auf true gesetzt, werden die nächsten zwei Attribute ignoriert.

*lastModFrom*: Dieses Attribut kann entweder auf openTime (der Standard) oder auf dirLastMod gesetzt werden. Der Wert openTime gibt an, dass die letzten Modifikationszeiten, verglichen mit dem Falls-Modifiziert-Seit header, welcher vom Client gesendet wurde, hinsichtlich der Zeit, ab welcher der Searcher startete berechnet werden sollte. dirLastMod sollte verwendet werden, falls man die Zeit auf das letzte Aktualisierungsdatum der Festplatte setzen möchte.

*etagSeed*: Der Wert dieses Attributs wird als der Wert des ETag Headers gesendet. Die Änderung dieses Werts kann dazu hilfreich sein, Kunden zu zwingen, erneut Inhalte abzurufen, selbst wenn die Indexe nicht verändert wurden, z.B. wenn Änderungen an der Konfiguration vorgenommen wurden.

```
<httpCaching      never304="false"
                  lastModFrom="openTime"
                  etagSeed="Solr">
  <cacheControl>max-age=30, public</cacheControl>
</httpCaching>
```

### cacheControl Element

Zusätzlich zu diesen Attributen, akzeptiert `<httpCaching>` das KindElement: `<cacheControl>`. Der Inhalt dieses Elements wird als der Wert des CacheControlHeaders mit HTTP Antworten gesendet. Der Header wird verwendet, um das Standardverhalten des Caches des anfragenden Clients zu ändern. Die möglichen Werte für den CacheControlHeader werden von der HTTP 1.1 Spezifikation definiert.

Das Einstellen des max-age Felds, steuert wie lange ein Client eine zwischengespeicherte Antwort wiederverwenden kann, bevor er sie erneut vom Server anfordert. Diese Zeitspanne sollte abhängig davon, wie oft man den Index updatet und ob es akzeptabel ist, veralteten Inhalt zu verwenden, gesetzt werden.

---

#### 3.1.6.6 REQUEST HANDLER UND SEARCH COMPONENTS

Nach dem `<query>` Abschnitt, werden die Request-Handler und die Suchkomponenten konfiguriert. Diese werden jeweils mit `<RequestHandler>` und `<SearchComponent>` eingeleitet. Ein RequestHandler verarbeitet die Anforderungen die zu Solr kommen. Eingehende Anfragen ("Queries") werden an den zuständigen Handler basierend auf dem Pfad in der Anfrage weitergeleitet. Dies kann beispielsweise "/select", dies ist der Standard Handler für Suchanfragen, oder "/suggest" sein. Der Standardhandler wird auch benutzt wenn bei der Anfrage kein Handler mit angegeben wurde. Eine Suchkomponente ist eine Funktion der Suche, wie Highlighting oder Facetten. Diese Suchkomponente wird in solrconfig.xml getrennt vom RequestHandler definiert und dann, wenn benötigt, mit einem RequestHandler verbunden.

Jeder RequestHandler wird mit einem Namen und einer Klasse definiert. Request Handler können mit dem Parameter "lazy" versehen werden, dass bedeutet er wird nicht geladen, bis er das erste Mal angesprochen wird.

Eine Standardanfrage kann man in der Abbildung 3-5 Standardsuchanfrage sehen.



Abbildung 3-5 Standardsuchanfrage

Der "q" Parameter wird immer benötigt, er gibt an nach was im Inhalt gesucht werden soll. Bei dieser Suche wird der Standardhandler "/select" benutzt, dieser ist folgendermaßen eingebunden.

```
<requestHandler name="/select" class="solr.SearchHandler"> <
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
    <str name="df">text</str>
  </lst>
</requestHandler>
```

Der "row" Parameter gibt an wie viele Suchergebnisse zurückgegeben werden sollen. Der "echo" Parameter gibt an, dass die in der Anfrage definierten Parameter im Falle einer debug Antwort mitgegeben werden.

Am Ende jeder RequestHandler Definition werden die "components" festgelegt. In diesem Abschnitt ist eine Liste mit SearchHandlers die zusammen mit dem RequestHandler genutzt werden kann. Sie werden nur über Request-Handler registriert.

Die Suchkomponenten definieren die Logik, die von den SearchHandler verwendet werden, um Abfragen für Benutzer durchzuführen. Es gibt mehrere Standardsuch Komponenten, die mit allen SearchHandlers arbeiten ohne das zusätzliche Konfiguration durchgeführt werden müssen (query, facet, mlt, highlight, stats, debug). Wenn keine anderen Komponenten definiert sind, werden diese standardmäßig verwendet.

Es ist möglich einige Komponenten vor (mit "first-components") oder nach (mit last-components") anderen Komponenten zu benutzen. Dies ist nützlich, wenn benutzerdefinierte Suchkomponenten konfiguriert werden sollen. Werden eigene Komponenten definiert werden die Standardkomponenten überschrieben.

### 3.1.6.7 VERÄNDERUNGEN

Da vor allem an der Erweiterung der Standardkonfiguration an die deutsche Sprache gearbeitet wurde, wurde an der Solr-Konfiguration selbst wenig verändert. Für zwei Elemente musste allerdings die Konfigurationsdatei angepasst werden, zum einen für Suggestions und zum anderen für die Facettierung.

#### Suggestion

Ein weiteres Feature welches der Suchservice anbietet ist Suggestion, dies wird auch Auto vervollständigung von Benutzereingaben genannt. Solr 3.1 hat einen Suggestor integriert, dieser musste an die Anforderungen angepasst werden. Die Vervollständigung ist durch einen neuen RequestHandler (/suggest) und eine neue Suchkomponente („suggest“) in der solrconfig.xml möglich (siehe Abbildung 3-6 Suggest Konfiguration).

```

<searchComponent name="suggest" class="solr.SpellCheckComponent">
  <lst name="spellchecker">
    <str name="name">suggest</str>
    <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
    <str name="lookupImpl">org.apache.solr.spelling.suggest.tst.TSTLookup</str>
    <str name="field">content_autocomplete</str>
    <str name="buildOnCommit">true</str>
  </lst>
</searchComponent>

<requestHandler name="/suggest" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <str name="spellcheck">true</str>
    <str name="spellcheck.count">10</str>
    <str name="spellcheck.maxCollations">10</str>
    <str name="spellcheck.maxCollationTries">1000</str>
    <str name="spellcheck.collateExtended"></str>
  </lst>
  <arr name="components">
    <str>suggest</str>
    <str>query</str>
  </arr>
</requestHandler>

```

Abbildung 3-6 Suggest Konfiguration

Der neue Suggester verwendet viel aus der SpellCheckComponent Infrastruktur, so verwendet er auch den gemeinsamen Rechtschreibprüfungsparameter „spellcheck = true“.

Es ist möglich die Suggester-Komponente mit weiteren Parametern konfigurieren:

**spellcheck.dictionary = suggest** – Das ist der Name der Wörterbuch-Komponente, die konfiguriert wird. Wird diese umbenannt, muss der Suggester in der Suggestanfrage mit dem neuen Namen angesprochen werden

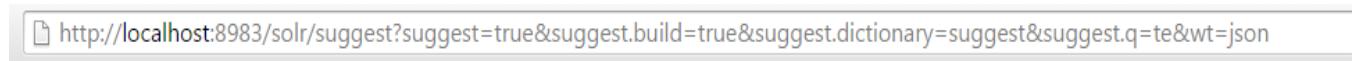
**spellcheck.onlyMorePopular = true** - Wenn dieser Parameter auf true gesetzt ist, so werden die Suggestions nach Gewicht ("Popularität") sortiert, ist dies auf false gesetzt, werden die Suggestions alphabetisch sortiert.

**spellcheck.count = 5** – Dieser Parameter legt fest, wie viele Vorschläge angezeigt werden sollen.

**spellcheck.collate = true** – Ist dieser Parameter aktiviert werden die ersten passenden Vorschläge gesammelt.

Die Vorschläge für die Vervollständigung werden aus einem Wörterbuch geliefert, das auf dem Hauptindex oder auf einem beliebigen anderen Wörterbuch basiert. Bei diesem implementierten Suggester wird es aus den bereits indizierten Inhalten erstellt. Der Parameter „field“ der Suchkomponente gibt an auf welchem Feld das Wörterbuch aufgebaut werden soll. Alle Wörter die in diesem Feld indiziert werden, werden auch ins Wörterbuch aufgenommen. Der Parameter "buildOnCommit" baut das Suggestwörterbuch, wenn er auf true eingestellt ist, bei jedem Commit. Es gibt zudem die Möglichkeit das Suggestwörterbuch

erst bei der Anfrage zu bilden, das wird beim Suchservice nicht benötigt, da es bei jedem Starten des SolrStack aufgebaut wird. Eine Anfrage bei der der Suggester erst gebildet wird, ist in Abbildung 3-7 Build Suchanfrage zu sehen.



```
http://localhost:8983/solr/suggest?suggest=true&suggest.build=true&suggest.dictionary=suggest&suggest.q=te&wt=json
```

Abbildung 3-7 Build Suchanfrage

Es wird ein neuer Feldtyp in der schema.xml benötigt, welcher die Informationen für Suggestions aufbereitet („content\_autocomplete“). In diesem werden die indizierten Inhalte für jedes Dokument gespeichert, zudem werden aus diesem Feld die einzelnen Wörter für das Suggestwörterbuch herausgearbeitet.

```
<field name="content_autocomplete" type="text_auto" indexed="true" stored="true" multiValued="false" />
```

Dieses Feld wird indiziert und auch gespeichert, kann aber nicht mehrere Inhalte beinhalten. Zudem basiert es auf dem Feldtypen „text\_auto“, welcher folgendermaßen definiert ist:

```
<fieldType class="solr.TextField" name="text_auto" positionIncrementGap="100" >
    <analyzer>
        <tokenizer class="solr.UAX29URLEmailTokenizerFactory"/>
        <filter class="solr.TypeTokenFilterFactory" types="types_blacklist.txt" useWhitelist="false"/>
        <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0" splitOnCaseChange="1"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.GermanNormalizationFilterFactory"/>
    </analyzer>
</fieldType>
```

Mit dieser Konfiguration von „text\_auto“ werden im Suggestwörterbuch nur einzelne Wörter gespeichert und als Autovervollständigung angeboten. Dies ist für diesen Suchservice praktikabler, denn ganze Phrasen eignen sich eher wenn Suggestion auf Titel und Überschriften angewandt werden.

Möchte man Phrasen unterstützen sollte folgende Konfiguration für „text\_auto“ verwendet werden.

```
<fieldType class="solr.TextField" name="text_auto" positionIncrementGap="100" >
    <analyzer>
        <tokenizer class="solr.KeywordTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
    </analyzer>
</fieldType>
```

Eine mögliche HTTP-Anfrage mit Autovervollständigung kann beispielsweise wie in Abbildung 3-8 Suggestanfrage aufgebaut sein.



Abbildung 3-8 Suggestanfrage

Die Bestandteile der Suchanfrage sind folgende:

**http://localhost:8983/solr/collection1/**: Mit diesem Abschnitt wird der richtige SolrServer angesprochen

**suggest**: Name des Request Handlers, welcher für Suggestions verantwortlich ist.

**?**: Dies wird zwischen RequestHandler und restlicher Anfrage an diesen gesetzt.

**spellcheck=true**: aktiviert Suggestions

**spellcheck.dictionary=suggest**: wird benötigt, damit das richtige Dictionary benutzt wird

**spellcheck.q=** : Nach „q“ müssen die zu vervollständigen Eingaben stehen, bei zwei getrennten Wörtern wird ein Leerzeichen mit "%20" eingefügt.

Eine mögliche Antwort (siehe Abbildung 3-9) von Solr, wenn nach „end“ gesucht wird

```
<?xml version="1.0" encoding="utf-16"?>
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
  </lst>
  <lst name="spellcheck">
    <lst name="suggestions">
      <lst name="end">
        <int name="numFound">3</int>
        <int name="startOffset">0</int>
        <int name="endOffset">3</int>
        <arr name="suggestion">
          <str>ende</str>
          <str>endlich</str>
        </arr>
      </lst>
    </lst>
  </result name="response" numFound="0" start="0">
</response>
```

Abbildung 3-9 Antwort bei einer Suggestanfrage

## Facettierung

Bei einer facettierten Suche können Suchergebnisse anhand verschiedener Kategorien weiter gefiltert werden. Datenelemente werden vom Indexer mit Schlagworten bzw. Kategorien versehen. Diese Schlagworte sind die Grundlage für die facettierte Suche. Bei den indizierten Dokumenten ist dies das Feld "cat", welches in der schema.xml definiert ist.

Die hierfür benötigten Einstellungen sind schon in der solrconfig.xml enthalten, sie werden durch Request-Handler angesprochen und geben einen „Facet counts“ Abschnitt mit der Suchantwort zurück. Der „Facet Counts“ Abschnitt enthält alle möglichen Facetten zur Suchanfrage. Es müssen lediglich bei der Suchanfrage die richtigen Parameter mitgegeben werden.

Damit die Facetten im Suchservice genutzt werden können, wird Field Faceting benötigt, so muss in der Anfrage der Parameter "facet" auf „true“ gesetzt werden. Mit "facet.field" kann eingeschränkt werden auf welchem der indizierten Felder der Dokumente die Kategorien aufgebaut werden sollen. Nun wird bei jeder Suchanfrage mitgeschickt wie viele der Kategorien auf der aktuellen Seite zur Verfügung stehen. Dann wird der Anfrage noch ein fq Parameter (Syntax: "Kategorienfeld:Parameter") hinzugefügt, dadurch können die möglichen zurückgegebenen Antworten verringert werden. Eine mögliche Anfrage mit Facetten ist in der Abbildung 3-10 zu sehen.

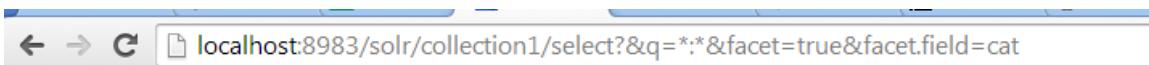


Abbildung 3-10 Anfrage mit Facetten

## 3.2 CLUSTERFÄHIGKEIT

Die Clusterfähigkeit ist ein wichtiger Aspekt in einem funktionierenden Suchservice. In diesem Abschnitt werden seine Komponenten und Konfiguration eingehend erläutert. Relevanz zeigt sich vor allem wenn es um Erreichbarkeit geht. Im Fall eines Absturzes einer Instanz müssen weiterhin lauffähige Komponenten vorhanden sein um kontinuierlich Suchanfragen bearbeiten zu können. Diese Fault-Tolerance kann gleichzeitig genutzt werden um eventuelle Übermengen an Suchanfragen auf die verschiedenen Clusterparts zu verteilen und somit ein Loadbalancing zu garantieren. Ein weiterer wichtiger Faktor ist die Antwortzeit des Servers, die durch vermehrte Ressourcen minimiert werden kann.

Diese Ressourcen werden allgemein durch große Serverfarmen erreicht, können aber ebenso durch virtuelle Clusterarchitekturen realisiert werden. Die Auswahl dafür verwendbarer Tools ist vielfältig und auf die eigenen Bedürfnisse abzustimmen.

### 3.2.1 SOLRCLOUD<sup>29</sup>

Als erste Testumgebung wurde SolrCloud verwendet. Die bereits in die Solr-Installation integrierte Cloudsoftware lässt sich ohne Umwege ausführen. Dafür wird ein Clustermanager namens „ZooKeeper“ verwendet, der sämtliche Einzelteile des Clusters, genannt „Shards“, kennt und ihnen Aufgaben zuweisen kann. Damit übernimmt dieser ZooKeeper die Aufgaben des Loadbalancing, der Fault-Tolerance und der Steuerung von SolrCloud. Eine Architektur des Clusters kann dann ähnlich wie in Abbildung 3-11: Solrcloud Architektur aussehen.

<sup>29</sup> <https://wiki.apache.org/solr/SolrCloud> (letzter Zugriff: 06.03.2015)

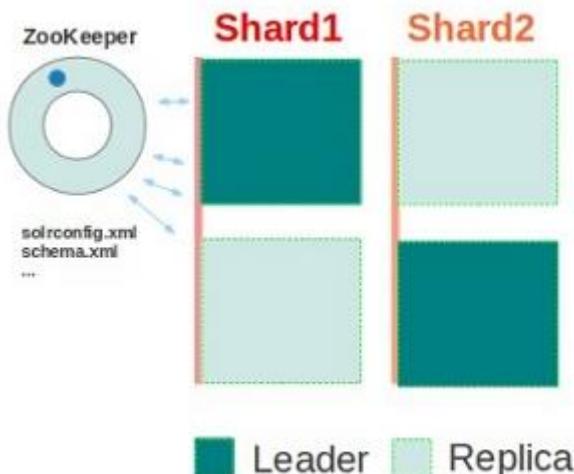


Abbildung 3-11: Solrcloud Architektur

Die hier genannten Replicas sind dabei Kopien von Leadern eines Shards. Sie sind eine automatisch gestartete Shard-Instanz, um alle Informationen redundant im Shard gespeichert zu halten, für den Fall das der Leader ausfällt, so dass kein Verlust von Datensätzen zu befürchten ist. Sämtlicher Datenverkehr wird dabei über den ZooKeeper auf die Clusterbereiche verteilt.

Eine Herausforderung ist dabei, dass SolrCloud nur ZooKeeper, die mit Shards verbunden sind, erstellt. Fällt der Shard aus, dem der einzige oder letzte ZooKeeper zugeordnet ist, so fällt auch die Such-Funktionalität aus.

Ein weiterer Nachteil ist nach einigen Tests aufgefallen: Der interne Clustermanager hat ein Limit bezüglich der Dateigrößen von Konfigurationsdateien. So wird Beispielsweise SolrCloud nicht erfolgreich ausgeführt, sobald eine Datei größer als 1 MB ist.

### 3.2.2 START DES SOLRCLOUD CLUSTERS<sup>30</sup>

Voraussetzung um SolrCloud zu verwenden ist eine korrekte Installation von Java und das Vorhandensein von Solr – die Example Datei sollte einmal ausgeführt worden sein.

#### 3.2.2.1 START VIA SKRIPT

SolrCloud wirbt mit der Einfachheit der Ausführung – so kann ein Cluster schon gestartet werden, sobald Solr auf dem Rechner einmal lokal getestet wurde. Durch das „Solr“-Script im „Bin“ Ordner kann man direkt einen Server mit integriertem ZooKeeper starten. Durch den Befehl

```
solr -e cloud
```

wird eine geführte Initialisierung eingeleitet (mit der Option -noprompt wird ein 2 Shard – 2 Replica Cluster auf den Standard-Ports aufgesetzt, ohne dass weitere Eingaben erforderlich sind).

<sup>30</sup> <https://cwiki.apache.org/confluence/display/solr/SolrCloud> (letzter Zugriff: 06.03.2015)

### 3.2.2.2 KONFIGURATION AN ZOOKEEPER ÜBERTRAGEN

Dabei werden die Konfigurationsdateien direkt an den ZooKeeper gesendet. Um dies manuell durchzuführen benutzt man das „zkCli“ Skript, welches sich im „scripts/cloud-scripts“-Ordner befindet.

```
zkcli.sh -cmd upconfig  
-zkhost <host:port>  
-confname <name for configset>  
-solrhome <solrhome>  
-confdir <path to directory with configset>
```

### 3.2.2.3 SOLRCLOUD MANUELL AUFSETZEN

Um ein Cluster komplett manuell aufzusetzen, sind spezielle Befehle nötig, welche im Ordner der auszuführenden Solr-Instanz zu starten sind.

```
java -Dbootstrap_confdir=<configuration directory>  
-Dcollection.configName=<collection name>  
-DzkRun -DnumShards=<number of shards>  
-jar start.jar
```

Dabei wird ein Solr-Server auf dem Standard-Port 8983 gestartet, welcher verbunden mit einer ZooKeeper-Instanz auf dem Port 9983 gekoppelt ist. Die Anzahl der Shards gibt dabei an, wie viele Shards erstellt werden, bevor alle neuen Instanzen als Replicas erstellt werden. Einen neuen Shard erstellt man durch den Befehl:

```
java -Djetty.port=<Shard/Replica Port>  
-DzkHost=<ZooKeeper URL>  
-jar start.jar
```

Dabei sollte jeder Shard in einem eigenen Ordner erstellt werden.

### 3.2.2.4 DATEN INDIZIEREN

Noch sind allerdings keine Daten auf dem Server vorhanden, nach denen gesucht werden könnte. Der einfache Befehl

```
java -Durl=http://<destination>:<PORT>/solr/collection1/update  
-jar post.jar <file>
```

kann Dateien indizieren. Dabei ist es unwichtig auf welchen Server man die Daten überträgt – sie gelangen automatisch dahin, wo sie gebraucht werden.

### 3.2.2.5 SOLRCLOUD ZURÜCKSETZEN

Um alle bisherigen Einstellungen zu löschen, werden einfach die solr/zoo\_data Order gelöscht.

### 3.2.2.6 BEISPIEL

Das manuelle Starten eines SolrCloud Servers auf dem eigenen System könnte also folgendermaßen stattfinden:

Schritt 1 – Erstellen eines Shards mit integriertem ZK (example)

```
java -Dbootstrap_confdir=./solr/collection1/conf  
      -Dcollection.configName=myconf -DzkRun  
      -DnumShards=2  
      -jar start.jar
```

Schritt 2 – Erstellen des zweiten Shards – verweist auf gerade erstellten ZK (example 2)

```
java -Djetty.port=7574  
      -DzkHost=localhost:9983  
      -jar start.jar
```

Schritt 3 – Wiederhole Schritt 2 um zusätzliche Replicas hinzuzufügen (andere Djetty.ports)

Schritt 4 – Daten indizieren (z.B. im exampledocs Ordner)

```
java -Durl=http://localhost:7574/solr/collection1/update  
      -jar post.jar ipod_video.xml  
java -Durl=http://localhost:8983/solr/collection1/update  
      -jar post.jar monitor.xml  
java -Durl=http://localhost:7574/solr/collection1/update  
      -jar post.jar mem.xml
```

Öffnet man dann im Browser [http://localhost:8983/solr/collection1/select?q=\\*>\\*](http://localhost:8983/solr/collection1/select?q=*>*), so wird die komplette Collection angezeigt.

### 3.2.2.7 API

Die Collections-API ist ein praktischer Teil von SolrCloud, um das Cluster auch noch nach dem Start zu konfigurieren. Welche Befehle dafür nötig sind, lässt sich in Tabelle 3-1: Cluster-API detailliert einsehen. Genauere Informationen dazu findet man hier: <https://cwiki.apache.org/confluence/display/solr/Collections+API>

<b>REST-Befehl</b>	<b>Funktion</b>
/admin/collections?action=CREATE	Create a collection
/admin/collections?action=RELOAD	Reload a collection
/admin/collections?action=SPLITSHARD	split a shard into two new shards
/admin/collections?action=CREATESHARD	Create a new shard
/admin/collections?action=DELETESHARD	Delete an inactive shard
/admin/collections?action=CREATEALIAS	create or modify an alias for a collection
/admin/collections?action=DELETEALIAS	delete or modify an alias for a collection
/admin/collections?action=DELETE	Delete a collection
/admin/collections?action=DELETEREPLICA	delete a replica of a shard
/admin/collections?action=ADDREPLICA	add a replica of a shard
/admin/collections?action=CLUSTERPROP	Add/edit/delete a cluster-wide property
/admin/collections?action=MIGRATE	Migrate documents to another collection
/admin/collections?action=ADDRULE	Add a specific role to a node in the cluster
/admin/collections?action=REMOVEROLE	Remove an assigned role
/admin/collections?action=OVERSEERSTATUS	Get status and statistics of the overseer
/admin/collections?action=CLUSTERSTATUS	Get cluster status
/admin/collections?action=REQUESTSTATUS	Get the status of a previous asynchronous request
/admin/collections?action=LIST	List all collections
/admin/collections?action=ADDREPLICAPROP	Add an arbitrary property to a replica specified by collection/shard/replica
/admin/collections?action=DELETEREPLICAPROP	Delete an arbitrary property from a replica specified by collection/shard/replica
/admin/collections?action=BALANCESHARDUNIQUE	Distribute an arbitrary property, one per shard, across the nodes in a collection

Tabelle 3-1: Cluster-API

### 3.2.3 ZOOKEEPER<sup>31</sup>

Der ZooKeeper kann auch extern gestartet werden. Der Vorteil davon ist, dass man mit einem lauffähigen ZooKeeper Ensemble die Sicherheit des Clusters erhöhen kann. Die Instanzen sind nicht mit Solr verknüpft und fällt ein Shard aus, so fällt nicht gleichzeitig ein Teil der Such-Funktionalität aus. Ein Nachteil ist die hohe Wartungsintensität. Bereits beim Aufsetzen des Systems ist deutlich höherer Aufwand zu betreiben.

#### 3.2.3.1 INSTALLATION UND KONFIGURATION

Zur Installation reicht es, ähnlich wie bei Solr, eine stabile Version herunterzuladen und darauf zu achten, dass Java aktuell ist. Dafür wurde die Version 3.4.6 von Apaches ZooKeeper genutzt. Über die im Bin-Ordner befindlichen Skripte lässt sich der Cluster-Manager dann starten. Dafür wird unter Windows einfach die ZkServer.cmd Datei ausgeführt – bei Linux basierten Betriebssystemen verwendet man im „Zookeeper“-Ordner den Befehl „bin/zkServer.sh start“ (zum Beenden wird natürlich „stop“ als Parameter verwendet).

Um Einstellungen vorzunehmen, kann man im „Conf“-Ordner die Zoo.cfg manipulieren.

Folgende Informationen sind dabei einstellbar:

tickTime :

Überprüft, ob alle Server noch am Laufen sind. Das minimale „Session Time Out“ ist dabei zwei Ticks lang. Die Zeit wird in Millisekunden angegeben.

dataDir:

In dem hier angegebenen Ordner werden die wichtigen Informationen bezüglich des Clusters gespeichert.

clientport:

Hier wird angegeben über welchen Port sich Solr mit dem ZooKeeper verbinden darf.

#### 3.2.3.2 ENSEMBLE

Für ein Ensemble werden mehrere Konfigurationsdateien benötigt. Um diese genauer beschreiben zu können wird nun ein Beispielinhalt aufgeführt:

```
tickTime=2000
dataDir=c:/sw/zookeeperdata/2
clientPort=2182
initLimit=5
syncLimit=2
server.1=localhost:2888:3888
server.2=vm1:2889:3889
server.3=vm2:2890:3890
```

<sup>31</sup> <http://zookeeper.apache.org/> (letzter Zugriff: 06.03.2015)

Die Werte von tickTime, dataDir und clientPort sind bereits erklärt. Dabei ist aber darauf zu achten, dass jede Konfigurationsdatei einen anderen Datenordner haben sollte. Auf verteilten Systemen geschieht das natürlich recht automatisch.

Die neuen Werte haben folgenden Nutzen:

initLimit:

Gibt die Anzahl an Ticks an, in der es erlaubt ist, mit einem Leader zu verbinden und zu synchronisieren. In diesem Fall wären es also 10 Sekunden, in denen der Server wartet, um sich zu verbinden.

syncLimit:

Gibt die Anzahl an Ticks an, die ein Folgeserver hat, um sich zu synchronisieren. Fällt einer zu weit zurück, so wird er fallen gelassen.

Server.X:

Zunächst: Das X wird durch eine ID ersetzt. Diese darf eine Zahl zwischen 1 und 255 sein. Wichtig ist, dass im „dataDir“-Ordner eine Datei ohne Dateiendung ist, welche als Inhalt die ID besitzt. Diese Datei muss den Namen „myid“ tragen. Die dahinter angegebene IP ist die Kommunikationsschnittstelle der jeweiligen Server.

Zum Start des Ensembles werden dann alle Konfigurationsdateien mit Hilfe des Startskripts ausgeführt. Unter Linux wird dafür an den „Start-Befehl“ noch die zu verwendende Konfigurationsdatei angehängt und sieht dann z.B. so aus:

```
bin/zkServer.sh start zoo1.cfg
```

Nach dem Start aller Server kann Solr an die genannten Clientports eingehängt werden. Dabei ist wichtig, dass alle IPs, die beim Start verwendet werden, eingefügt werden. Dies geschieht im DzkHost- Parameter mit Kommata verknüpft.

### 3.3 WEBSERVICE

Der Webservice ist die selbst entwickelte Komponente, welche zur Kommunikation mit dem SolrStack verwendet wird. Die Kommunikation erfolgt über eine vom Webservice zur Verfügung gestellte REST-API. Die sich daraus ergebenen Anforderungen sind im Folgenden aufgeführt.

#### 3.3.1 ANFORDERUNGEN

Die im Mittelpunkt stehende Anforderung ist die Bereitstellung der REST-API. Über diese sollen Anfragen für den SolrStack eingehen, Befehle angenommen und Informationen verarbeitet werden.

Um dieser Anforderung zu entsprechen, muss der Webservice einen Webserver zur Verfügung stellen. Die Wahl ist hierbei auf Jetty<sup>32</sup> gefallen, da Jetty ein leistungsstarker Server mit wenigen unnötigen Funktionen ist, der problemlos in eine Java Applikation integrierbar ist. Um die REST-Anfragen anzunehmen waren zwei Ansätze in der Diskussion.

- Normale HttpServlets<sup>33</sup>
- Jersey<sup>34</sup>

Nachdem zuerst HttpServlets verwendet wurden, wurde nach kurzer Zeit jedoch auf Jersey umgestiegen, da HttpServlets deutlich mehr Code für die gleiche Funktionalität benötigen. Auch sind viele Standardanforderungen in Jersey schon vorimplementiert, weswegen mehr Zeit für projektspezifische Probleme zur Verfügung stand.

Zusätzlich zu der REST-API ist eine Webseite benötigt, über die in einer Google-ähnlichen Ansicht die REST-API ansprechbar sein muss. Um diese zu verwirklichen, wird der gleiche Jetty Webserver genutzt. Die Seiten werden jedoch mithilfe von HTML, JavaScript und CSS dargestellt. Als Framework dient Bootstrap<sup>35</sup>, da hiermit auf eine große Sammlung von Elementen zugegriffen werden kann.

Neben dieser zentralen Anforderung wird eine einfache und stabile Verbindung zum SolrStack benötigt. Zwar lässt sich der SolrStack über eine REST-API ansprechen, jedoch gibt es auch eine einfachere Lösung: SolrJ<sup>36</sup>. Durch den Zugriff über SolrJ muss sich nicht um das Versenden von HttpServletRequest und das Auswerten von Antworten gekümmert werden, sondern es kann der SolrStack über ein ganz normales Java Objekt mit vordefinierten Methoden angesprochen werden. Auch ist es durch die Verwendung von SolrJ möglich mit einem Cluster als auch mit einem einzelnen Knoten auf die gleiche Art und Weise zu kommunizieren.

Da das Projekt so einfach wie möglich auf virtuellen Maschinen bzw. auf einzelnen eigenen Rechner laufen soll, ist eine Abhängigkeitsverwaltung benötigt: Maven<sup>37</sup>. Durch Maven ist es möglich, das Projekt ohne Probleme in eine IDE (wie zum Beispiel eclipse<sup>38</sup>) zu importieren, ohne die hohe Anzahl an Abhängigkeiten (unter anderem Jetty, Jersey, SolrJ, ...) manuell hinzuzufügen. Stattdessen lädt nach dem Importieren des Projektes Maven alle benötigten Abhängigkeiten und man ist sofort in der Lage den Webservice auszuführen bzw. weiterzuentwickeln.

Um nicht immer das Projekt aus einer IDE heraus starten zu müssen, ist eine Alternative benötigt. Der intuitivste Weg um ein in Java geschriebenes Programm zu starten, ist die Verwendung von Jar Dateien. Viele Betriebssysteme erlauben es sogar durch einen einfachen Doppelklick eine Jar auszuführen. Da jedoch alle Bibliotheken in der Jar gespeichert werden

---

<sup>32</sup> <http://www.eclipse.org/jetty/> (letzter Zugriff: 01.03.2015)

<sup>33</sup> <http://www.servlets.com/javadoc/javax/servlet/http/HttpServlet.html> (letzter Zugriff: 05.03.2015)

<sup>34</sup> <https://jersey.java.net/> (letzter Zugriff: 02.03.2015)

<sup>35</sup> <http://getbootstrap.com/> (letzter Zugriff: 28.02.2015)

<sup>36</sup> <http://wiki.apache.org/solr/Solrj> (letzter Zugriff: 05.03.2015)

<sup>37</sup> <https://maven.apache.org/> (letzter Zugriff: 01.03.2015)

<sup>38</sup> <http://eclipse.org/> (letzter Zugriff: 08.03.2015)

sollen, um die Nutzung möglich leicht zu machen, ist das maven-assambly-plugin<sup>39</sup> gefordert. Dieses ermöglicht es das Bauen der Jar zu automatisieren und zu konfigurieren. Das Endergebnis ist eine Jar Datei, in welcher der Quellcode und die externen Bibliotheken vollständig enthalten sind.

### 3.3.2 LAYOUT

Der Webservice hat zwei zentrale Bestandteile. Als erstes ist die Solr Klasse zu nennen, welche mithilfe von SolrJ mit dem SolrStack kommuniziert. Diese Solr Klasse erhält ihre Instruktionen über die zweite Komponente: Die REST-API. Die REST-API ist eine Sammlung von Jersey-Servlets, welche über bestimmte URLs erreichbar sind. Sie wird auch von der selbst entwickelten Webseite genutzt, um den SolrStack zu benutzen.

Zwar ist der Webservice schon weitgehend vorkonfiguriert, allerdings kann es nötig sein, bestimmte Parameter manuell zu verändern. Dies ist in der Datei config.properties möglich, die unter dem Pfad src/main/resources zu finden ist (bzw. in der kompilierten JAR-Datei) und deren Konfiguration beim Starten des Webservices eingelesen wird.

Folgende Parameter können angepasst werden:

- der Port, auf dem der Webservice erreichbar sein soll
- die URL zum SolrStack (Solr-Server)
- der Modus, in welchem der SolrStack gestartet wird (Cluster oder Lokal)

Es ist jedoch nicht unbedingt notwendig die config.properties anzupassen, wenn sich ein Parameter geändert haben sollte. Wenn man dies nicht wünscht, besteht die Möglichkeit, den Webservice über die Kommandozeile zu starten und dabei Parameter zu übergeben:

Die Parameter können genutzt werden um den Startmodus (im Folgenden nur "Modus", entweder *local* oder *cluster*) und die URL für den zu verbindenden SolrStack bzw. den Zookeeper (im folgenden nur "URL") festzulegen, nur der Port ist definitiv in der config.properties zu definieren.

Dabei ist zwischen folgenden Fällen zu unterscheiden:

#### Start ohne Parameter

Der Modus und die URL werden aus der config.properties übernommen.

Beispielauftrag in der Konsole: `java -jar webservice.jar`

#### Start mit einem Parameter

Der Parameter muss entweder local oder cluster sein. Legt fest, in welchem Modus gestartet wird. Die URL wird dann passend aus der config.properties genutzt.

Beispielauftrag in der Konsole: `java -jar webservice.jar local`

---

<sup>39</sup> <https://maven.apache.org/plugins/maven-assembly-plugin/> (letzter Zugriff: 03.03.2015)

## Start mit zwei Parametern

Der erste Parameter muss entweder local oder cluster sein. Legt fest, in welchem Modus gestartet wird.

Der zweite Parameter muss die URL sein.

Beispielauftrag in der Konsole: `java -jar webservice.jar local http://localhost:8983/solr`

### 3.3.3 SOLR KLASSE

Die Solr Klasse fungiert praktisch als Schnittstelle zwischen dem SolrStack und der REST-API. In ihr werden sowohl Methoden definiert, die von außen mithilfe der REST-API angesprochen werden können um mit dem SolrStack zu kommunizieren, als auch Hilfsmethoden, die für die interne Verarbeitung von Bedeutung sind. Je nach Startmodus (lokal oder cluster) wird intern beim initialisieren entweder eine Verbindung zu einem HttpSolrServer<sup>40</sup> oder einem CloudSolrServer<sup>41</sup> hergestellt werden. Nach Außen ergibt sich kein Unterschied in der Benutzung.

Alle Methoden der Solr Klasse bereiten nur die von der REST-API erhaltenen Parameter (siehe Abschnitt REST-API) auf, stellen eine Anfrage mittels SolrJ an den SolrStack und leiten die Antwort an die REST-API zurück.

Folgende Methoden sind in der Solr Klasse enthalten:

- `performQuery`: Wird angesprochen, wenn eine Suchanfrage durchgeführt oder eine Suggestion angefordert wird.
- `addDocumentsFromXML`: Wird angesprochen, wenn die CRUD Operation *Create* durchgeführt wird.
- `getDocument`: Wird angesprochen, wenn die CRUD Operation *Read* durchgeführt wird.
- `editFieldValues`: Wird angesprochen, wenn die CRUD Operation *Update* durchgeführt wird.
- `deleteDocument`: Wird angesprochen, wenn die CRUD Operation *Delete* durchgeführt wird.
- `indexFile`: Wird angesprochen wenn eine Datei indiziert wird.
- `documentExists`: Wird von der REST-API benutzt, um zu prüfen ob ein Dokument mit dieser ID vorhanden ist.

Im Folgenden wird beispielhaft die Implementierung der `getDocument`-Methode gezeigt (siehe Abbildung 3-12: `getDocument`-Methode).

---

<sup>40</sup> [https://lucene.apache.org/solr/4\\_3\\_1/solr-solrj/org/apache/solr/client/solrj/impl/HttpSolrServer.html](https://lucene.apache.org/solr/4_3_1/solr-solrj/org/apache/solr/client/solrj/impl/HttpSolrServer.html)

<sup>41</sup> [https://lucene.apache.org/solr/4\\_5\\_0/solr-solrj/org/apache/solr/client/solrj/impl/CloudSolrServer.html](https://lucene.apache.org/solr/4_5_0/solr-solrj/org/apache/solr/client/solrj/impl/CloudSolrServer.html)

```

public SolrDocument getDocument(String docId) {
    SolrQuery query = new SolrQuery();
    //Alle Documente mit der übergebenen docId
    query.setQuery("id:" + docId + "");

    QueryResponse response = null;
    try {
        //Suchanfrage ausführen
        response = solrStack.query(query, METHOD.GET);
    } catch (SolrServerException e) {
        e.printStackTrace();
    }

    //Ergebnis liefern, falls keine Übereinstimmung null
    if (response.getResults().getNumFound() != 0)
        return response.getResults().get(0);

    return null;
}

```

Abbildung 3-12: getDocument-Methode

### 3.3.4 REST-API<sup>42</sup>

REST ist ein Akronym und steht für REpresentational State Transfer. Eine REST-API stellt ein einheitliches Interface zur Verfügung, über das Ressourcen (über eine URI) angesprochen werden können bzw. deren Repräsentation (in verschiedenen Formaten, wie JSON oder XML).

Die Art des Zugriffs erfolgt dann über die verwendete Methode (GET, POST, PUT,...) und ggf. mitgeschickter Daten (als Parameter oder im Body) in der HTTP-Anfrage. Eine Anfrage ist immer zustandslos, so dass jede Anfrage alle erforderlichen Daten enthält, um die entsprechende Aktion auf dem Server auszuführen. Der Server liefert als Antwort einen HTTP-Statuscode und – je nach Anfrage – einen Inhalt zurück.

In den folgenden Abschnitten werden das Framework Jersey, welches zur Umsetzung der REST-API verwendet wurde, sowie die damit implementierten Schnittstellen kurz vorgestellt und erläutert.

#### 3.3.4.1 UMSETZUNG EINES RESTFUL WEBSERVICES MIT JERSEY

Bei Jersey<sup>43</sup> handelt es sich um ein Framework, welches die JAX-RS-Spezifikation implementiert, also die Java API for RESTful Web Services. Diese vereinheitlicht und ermöglicht die Verwendung des REST-Architekturstils in Webservices.

In diesem Projekt wird Jersey in der Version 2.16 eingesetzt, um die Entwicklung der REST-Schnittstelle zu vereinfachen. So können Java-Methoden in einer Klasse, die später als Servlet ausgeführt wird, entsprechend des REST-Prinzips Anfragen von einem Client annehmen und

<sup>42</sup> <http://www.RESTapitutorial.com/lessons/whatisREST.html> (letzter Zugriff: 01.03.2015)

<sup>43</sup> <https://jersey.java.net> (letzter Zugriff: 05.03.2015)

Antworten zurückliefern. Des Weiteren kann die Umwandlung von POJOs (Plain Old Java Objects) in JSON oder XML automatisiert erfolgen, so dass der Programmieraufwand entsprechend reduziert wird.

Zur Veranschaulichung folgt ein simpler Code-Ausschnitt, der den Zugriff auf Dokumente im Solr-Index ermöglicht, aus der implementierten REST-API.

```
@Path("/document")
public class DocumentResource {

    @GET
    @Path("/{docId}")
    @Produces("application/json" + ";charset=utf-8")
    public Object getDocument(@PathParam("docId") String docId) {

        docId = getDecodedId(docId);
        SolrDocument doc = Mastermind.solr.getDocument(docId);

        Object response = (doc == null) ? Response.status(404).build() : doc;
        return response;
    }

    @POST
    @Consumes("application/xml")
    public Response addDocument(String xml) {
        return Mastermind.solr.addDocumentsFromXml(xml);
    }
}
```

Abbildung 3-13: Codeausschnitt DocumentResource.java

Die gezeigten Methoden implementieren das Lesen und Erstellen eines Dokuments und sind beide über den Pfad */document* erreichbar, wobei für den Aufruf der *getDocument*-Methode zusätzlich die Id eines Dokuments angehängt werden muss: */document/id*. Dies wurde durch die *@Path*-Annotation festgelegt. Innerhalb der Methode wird dann der eigentliche Programmcode ausgeführt und das Ergebnis mit Statuscode an den aufrufenden Client zurückgegeben.

Auch ist ersichtlich, über welche HTTP-Methode die einzelnen Methoden aufgerufen werden müssen, nämlich GET und POST und welche Dateiformate zurückgeliefert bzw. erwartet werden (JSON bzw. XML).

Die *Mastermind.solr.getDocument*-Methode liefert nur ein Objekt vom Typ SolrDocument zurück. Jersey übernimmt hier die automatische Umwandlung in eine JSON-Repräsentation, um das sich der Programmierer aufgrund der *@Produces*-Annotation in diesem Fall nicht selbst kümmern muss.

Im Folgenden werden nun die einzelnen Schnittstellen vorgestellt, die für dieses Projekt implementiert wurden. Angegebene Standardwerte wurden direkt in der Schnittstelle definiert (über die Annotation *@DefaultValue*).

### 3.3.4.2 SUCHANFRAGE DURCHFÜHREN

**URL:** rest/request

**Method:** GET

#### Required Parameters:

q [String]

- Suchanfrage. Kann aus beliebig vielen Wörtern bestehen, eine Filterung ist möglich (z.B. nach Feld, wie url:http://www.tu-berlin.de)
- Beispiel: ?q=**Fakultät%204%20Lehrstuhl%20ISE**

#### Optional Parameters:

rh [String]

- Der RequestHandler.
- Standartwert: *select*
- Beispiel: ..&rh=select

start [int]

- Startwert des ersten Suchergebnisses, das zurückgegeben wird
- Standartwert: 0
- Beispiel: ..&start=0

rows [int]

- Beschränkung der zurückgegebenen Suchergebnisse, beginnend bei *start*
- Standartwert: 99999
- Beispiel: ..&rows=10

fq [String]

- Der FilterQuery-String, zur Filterung des Suchergebnisses nach Facetten; werden mit Komma getrennt
- Beispiel: ..&fq=Lehrstuhl,ISE

hl [boolean]

- Angabe, ob Highlighting (mitsenden von Textausschnitten für jedes Suchergebnis, in dem Begriffe hervorgehoben wurden) verwendet werden soll.

Success Response:

Code: 200

- Content: JSON-Repräsentation von ResultObject:
- numFound: Die Anzahl der Suchergebnisse insgesamt
- start : Der start-Wert (siehe oben)
- rows : Der rows-Wert (siehe oben)
- result : Die Ergebnisse der Suche (JSON von SolrDocumentList)

- facetResult : alle Facetten und deren Häufigkeit (bezogen auf alle Ergebnisse)
- highlighting : Snippets mit Highlighting für jedes gefundene Dokument (JSON Array)

Code: 204

- Anfrage war erfolgreich, aber es wurden keine Suchergebnisse gefunden.

### Error Response:

Code: 500

- Verbindung zum SolrStack fehlgeschlagen.

Zum Stellen einer Suchanfrage über die REST-API genügt lediglich das Setzen des Parameters *q*. Alle weiteren Parameter sind optional und deren Bedeutung der obigen Aufstellung zu entnehmen. Die API löst intern einen Aufruf der *Mastermind.solr.performQuery*-Methode aus, welche wiederum mittels der übergebenen Parameter eine Anfrage an den SolrStack stellt. Abhängig davon werden Daten an die API übermittelt und im JSON-Format an die aufrufende Anwendung ausgeliefert.

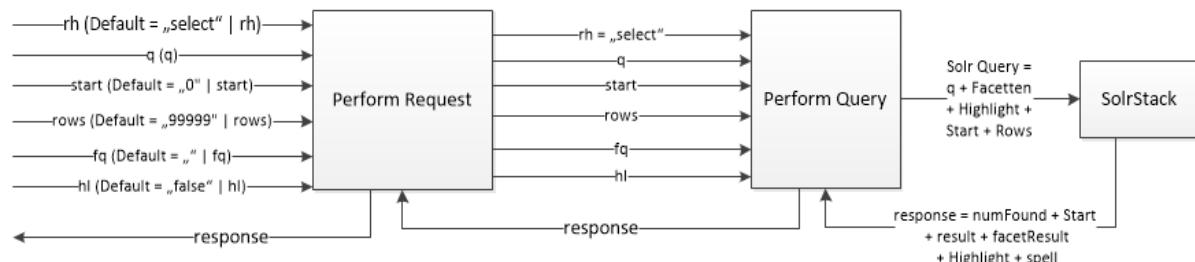


Abbildung 3-14: Suchanfrage stellen

#### 3.3.4.3 AUTOMATISCHE VERVOLLSTÄNDIGUNG

Um Vorschläge für einen Suchbegriff (nachfolgend „Suggestions“ genannt) anzufordern, wird die gleiche API-Schnittstelle genutzt, wie bei der normalen Suchanfrage. Allerdings ist hier der optionale Parameter *rh* auf *suggest* zu setzen. Der Parameter *q* beinhaltet den Begriff, für den die Suggestion angefordert wird. Zwar hat die Schnittstelle wie oben aufgeführt noch mehr optionale Parameter, diese werden allerdings ignoriert (siehe dazu auch Abbildung 3-15) noch einmal veranschaulicht. „Perform Request“ stellt hierbei die REST-API-Schnittstelle dar und „Perform Query“ ist die Methode in der Solr-Klasse, welche alle Parameter - außer *rh* und *q* - verwirft.

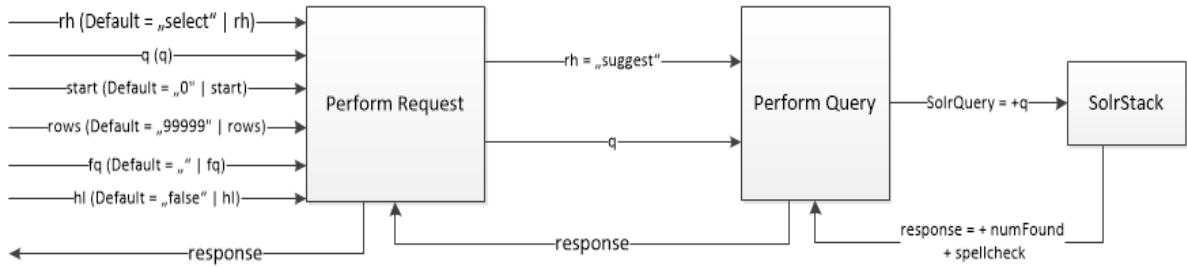


Abbildung 3-15 - Suggestions anfordern

### 3.3.4.4 CRUD FÜR DOKUMENTE

Das Akronym **CRUD** steht für **C**reate, **R**ead, **U**pdate, **D**elete und umfasst damit die grundlegenden Datenbankoperationen für das Erstellen, Lesen, Aktualisieren und Löschen von Datensätzen.

Diese elementaren Funktionen wurden ebenfalls für das Indizieren von Dokumenten in Solr implementiert und können über die REST-API genutzt werden.

#### 3.3.4.4.1 DOKUMENT ERSTELLEN (CREATE)

**URL:** rest/document/  
**Method:** POST  
**Content-Type:** application/xml

#### Http Body:

```

<add>
  <doc>
    <field name="id">id1</field>
    <field name="content">Lorem Ipsumi</field>
    <field name="url">http://www.example.de/</field>
  </doc>
  <doc>
    <field name="id">id2</field>
    <field name="content">Dolor sit.</field>
    <field name="url">http://www.example.de/amet</field>
  </doc>
</add>

```

Abbildung 3-16: Zwei zu indexierende Dokumente im XML-Format

#### Success Response:

Code: 201

- Dokument wurde erstellt, Rückgabe der location(s) im Header bzw. Body (bei mehreren erstellten Dokumenten)

#### Error Response:

Code: 500

- Serverfehler (z.B. XML enthält nicht alle als *required* markierten Felder).

Um ein Dokument zu erstellen, wird im Body der HTTP-Anfrage ein XML-Text mitgeschickt, der dem obigen Format entspricht. Intern wird dazu das XML verarbeitet und in ein bzw. mehrere Objekte vom Typ SolrInputDocument umgewandelt, die dann dem SolrStack übergeben und anschließend indiziert werden. Existierende Dokumente mit gleicher ID werden vollständig überschrieben. In der Antwort werden die URLs der neu erstellten Dokumente zurückgeliefert.

#### 3.3.4.4.2 DOKUMENT LESEN (READ)

---

**URL:** rest/document/<id>

**Method:** GET

##### Success Response:

Code: 200

- Ausgabe des Dokuments als JSON

```
{  
    "id": "id1",  
    "content": "Lorem Ipsum!",  
    "content autocomplete": "Lorem Ipsum!",  
    "url": "http://www.example.de/",  
    "_version_": 1494594113995014100  
}
```

Abbildung 3-17: Beispieldokument im JSON-Format

##### Error Response:

Code: 404

- Dokument existiert nicht

Die in der URL übergebene <id> des angefragten Dokuments wird intern beim SolrStack angefragt. Existiert dieses Dokument, wird es an die Schnittstelle übergeben und dabei in ein JSON-Format umgewandelt und vom Server als Antwort zurückgeliefert.

**Zu beachten:** Der im Projekt verwendete Webcrawler (siehe Abschnitt „Nutch“) setzt die ID eines Dokuments immer auf dessen Ursprungs-URL. Soll ein solches Dokument bearbeitet werden, muss die URL zunächst encodiert<sup>44</sup> werden und kann anschließend als ID innerhalb der URL des Aufrufes verwendet werden. Dies ID wird intern dann wieder decodiert.

#### 3.3.4.4.3 DOKUMENT BEARBEITEN (UPDATE)

---

**URL:** rest/document/<id>

**Method:** PUT

##### Required Parameters:

field [String]

---

<sup>44</sup> <http://de.wikipedia.org/wiki/URL-Encoding> (letzter Zugriff: 02.03.2015)

- Das Feld, das bearbeitet werden soll.

modifier [String]

- set (ersetze alle vorhandenen Werte im Feld)
- add (füge die übergebenen Werte hinzu)
- remove (lösche die übergebenen Werte aus dem Feld)

values [String]

- Werte, jeweils durch Komma getrennt

#### **Success Response:**

Code: 204

Dokument wurde entfernt.

#### **Error Response:**

Code: 404

Dokument existiert nicht.

Das Bearbeiten eines Dokumentes geschieht im eigentlichen Sinne immer über das Bearbeiten eines Feldes innerhalb des Dokumentes im Index. Dazu werden der Name des Feldes (z.B. cat), die zu bearbeitenden Werte und der „modifier“ gesetzt, der entscheidet, welche Aktion durchgeführt werden soll. Hierbei wird ein „Atomic Update“ durchgeführt, welches Solr von Hause aus unterstützt. Dabei werden nur die übergebenen Werte im vorhandenen Dokument eingefügt gelöscht oder bearbeitet.<sup>45</sup> Es wird also zunächst überprüft, ob ein Dokument mit der übergebenen ID im Index existiert und dann das Update ausgeführt.

```
SolrInputDocument doc = new SolrInputDocument();
doc.addField("id", docId);

Map<String, Object> partialUpdate = new HashMap<String, Object>();
partialUpdate.put(modifier.toString(), values);
doc.addField(field, partialUpdate);

solrStack.add(doc);
solrStack.commit();
```

Abbildung 3-18: Codeausschnitt atomares Update

#### 3.3.4.4.4 DOKUMENT LÖSCHEN (DELETE)

**URL:** rest/document/<id>

**Method:** DELETE

---

<sup>45</sup> <http://heliosearch.org/solr/atomic-updates/> (letzter Zugriff: 04.03.2015)

**Success Response:**

Code: 204

- Dokument wurde gelöscht

**Error Response:**

Code: 404

- Dokument existiert nicht

Es wird zunächst geprüft, ob das Dokument mit der übergebenen ID existiert. Ist dies der Fall, wird das Dokument mit der entsprechenden ID vollständig aus dem Index gelöscht.

---

**3.3.4.5 DATEI INDIZIEREN**

**URL:** rest/file/index

**Method:** POST

**Optional Parameters:**

replace [boolean]

- Gibt an, ob eine Datei ersetzt werden soll, wenn diese bereits indiziert wurde.  
Standartwert: *false*

location [String]

- Der Ort, an dem die Datei auffindbar sein soll (beispielsweise eine URL). Wird keine *location* angegeben, wird die Datei im servereigenen Dateisystem gespeichert.

**Http Body:**

Beinhaltet eine Datei. Gesandt mit multipart/form-data.

**Success Response:**

Code: 200

Content: Log des Vorganges.

**Error Response:**

Code: 302

Content: Log des Vorganges

- Die Datei existiert bereits, aber *replace* war *false*. Es wurde keine Indexierung vorgenommen.

Nach Eintreffen der Anfrage werden einige Gegebenheiten überprüft. Dieser Ablauf ist in Abbildung 3-19 dargestellt. Zunächst wird überprüft, ob der optionale Parameter *location* gesetzt ist. Falls dies nicht der Fall ist, soll die Datei im Dateisystem gespeichert werden. Der *URL* Parameter des Solr Dokumentes wird für diesen Zweck auf eine URL gesetzt, welche auf die Datei Laden-Schnittstelle (siehe 3.3.4.6) dieser REST-API verweist. Wichtig: Wenn

der Webservice nicht lokal genutzt wird, muss im Anschluss die URL noch manuell über die REST-API auf die korrekte URL gesetzt werden, da die automatisch Ermittelte in diesem Falle nicht funktioniert. Somit können auch Dateien, die nicht über einen Webserver zur Verfügung stehen indiziert, gesucht und bei Bedarf geladen werden. Falls *location* gesetzt ist, muss die Datei nicht im Dateisystem gespeichert werden und *URL* wird auf die übergebene *location* gesetzt.

Der nächste Schritt ist die Überprüfung, ob schon eine Datei mit dieser ID (die ID ist der übergebene Dateiname) schon vorhanden ist. Falls dem nicht so ist, kann zum nächsten Schritt vorangeschritten werden. Sollte die Datei jedoch vorhanden sein, wird überprüft, ob der übergebene *replace* Parameter *true* oder *false* ist. Bei *false*, wird der Status Code 302 zurückgegeben, da das Überschreiben nicht gewünscht ist. Weder der SolrStack noch das Dateisystem wird dadurch geändert. Falls *replace* auf *true* ist, wird die Datei überschreiben. Der letzte Schritt vor dem Absenden an Solr ist die Setzung des Timestamps (Zeitstempel), welcher den Zeitpunkt der letzten Änderung beinhaltet. Sollte all dies erfolgreich gewesen sein, gibt die REST-Schnittstelle den Statuscode 200 zurück.

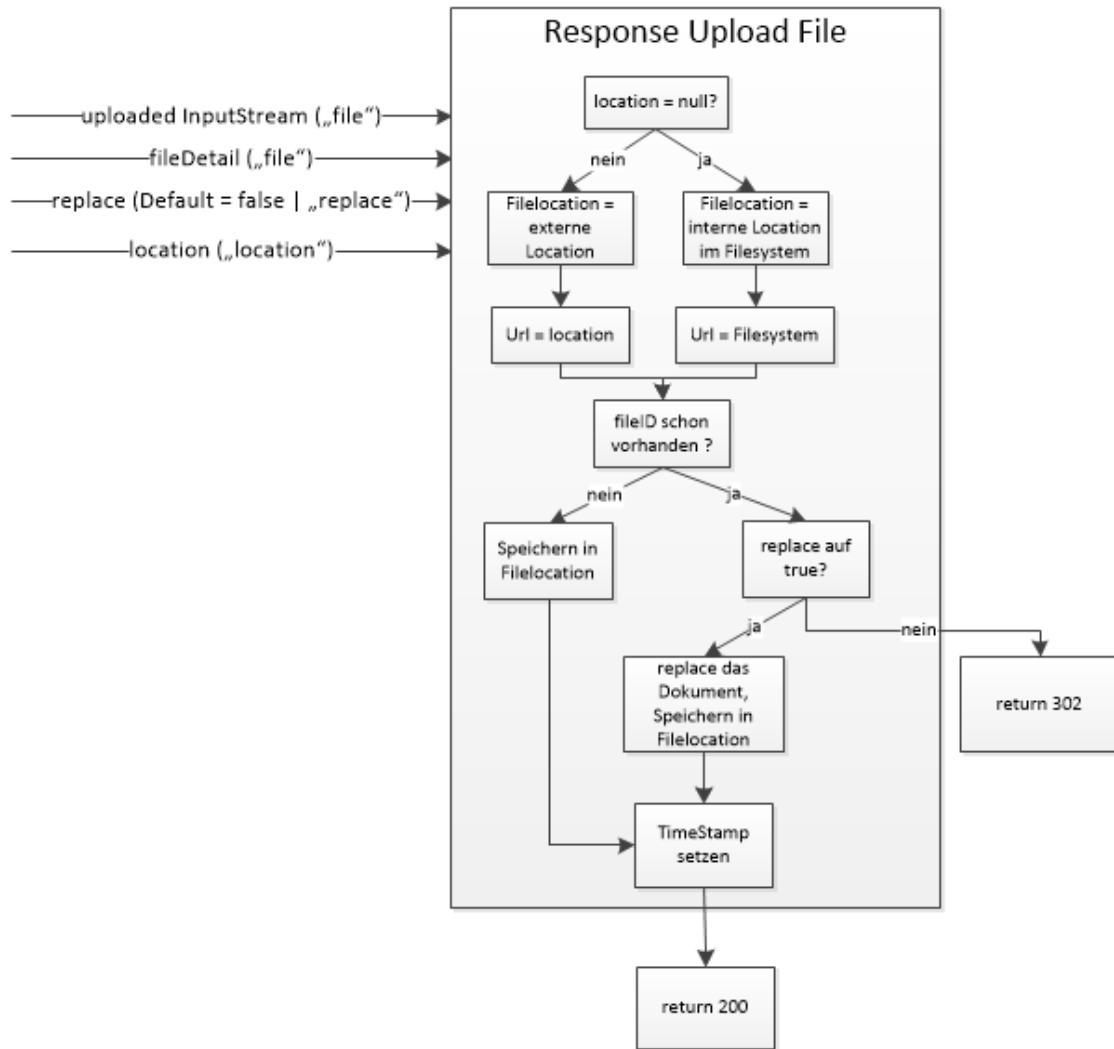


Abbildung 3-19: Datei indizieren

### 3.3.4.6 DATEI LADEN

**URL:** rest/file/download

**Method:** GET

#### Required Parameters:

id [String]

- ID der Datei, der Datei, die geladen werden soll.

#### Success Response:

Code: 200

- Content: Datei als application/octet-stream. Der Dateiname ist die ID.

#### Error Response:

Code: 404

- Es existiert keine Datei mit dieser ID im Dateisystem.

Benötigt wird diese Schnittstelle, um nach der Indexierung von Dateien, die nicht über einen Webserver erreichbar sind, einen Zugriff auf diese Dateien zu erhalten. Der Ablauf ist in Abbildung 3-20 dargestellt. Falls eine Datei mit der übergebenen ID existiert, wird diese als application/octet-stream zusammen mit dem HTTP Statuscode 200 zurückgegeben. Falls keine Datei mit dieser ID existiert, wird der Code 404 zurückgegeben.

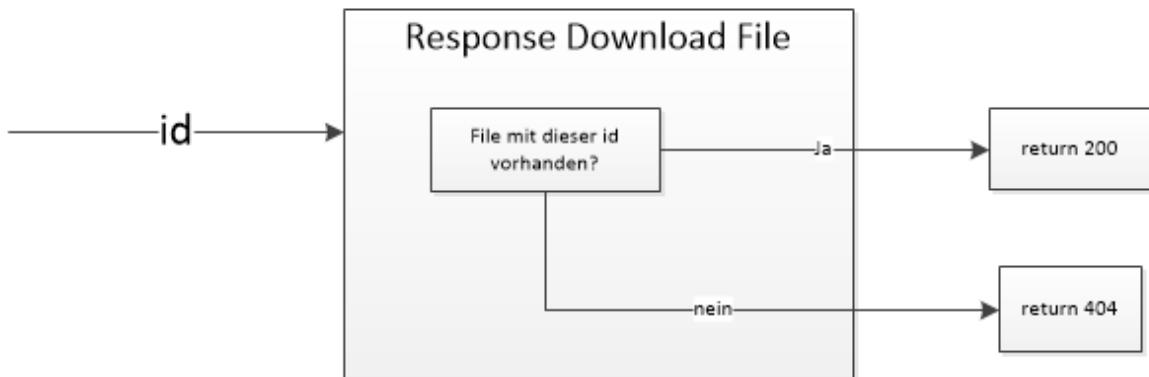


Abbildung 3-20 - Datei laden

### 3.3.5 WEBOBERFLÄCHE

Als Bestandteil des Projekts wurde eine beispielhafte Weboberfläche entwickelt, welche über die implementierte REST-API auf den Webservice zugreift und die zurückgelieferten Ergebnisse entsprechend aufbereitet präsentieren und dem Nutzer eine komfortable Filterung der Ergebnisse ermöglichen soll. Außerdem können die Ergebnisse durchgeblättert und über die Suggestion-Funktion Nutzereingaben vervollständigt werden.

Da der Fokus eher auf der eigentlichen Konfiguration von Solr und dessen Komponenten, dem Indizieren und Bearbeiten von Inhalten via Grabber bzw. der zu implementierenden REST-API lag, ist die Logik der Weboberfläche der Einfachheit halber komplett clientseitig über JavaScript (bzw. das verwendete Framework jQuery) realisiert worden.

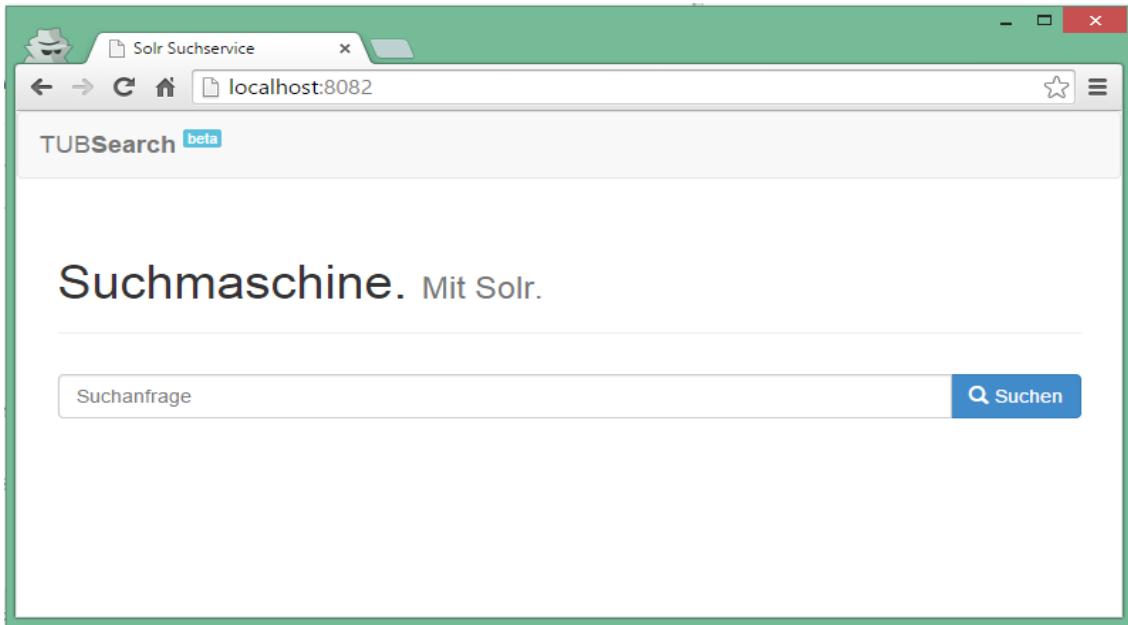


Abbildung 3-21: Eingabemaske

### 3.3.5.1 SUCHEN

Mit einem Klick auf „Suchen“ wird die Suchwort-Kombination im Eingabefeld mittels AJAX (Asynchronous JavaScript and XML) an die REST-Schnittstelle übermittelt:

```
Request URL: http://localhost:8082/rest/request?q=Student&start=0&rows=10&fq=&hl=true  
Request Method: GET
```

Abhängig von der Antwort des Servers, wird die Oberfläche aktualisiert:

- Status Code: 200 OK

Der Server liefert die gefundenen Ergebnisse im JSON-Format zurück. Einzelne Elemente daraus (Anzahl der Ergebnisse, gefundene Facetten inkl. Anzahl, eigentliche Ergebnisse und Textausschnitte mit Hervorhebungen des Suchbegriffs) werden extrahiert und in der Oberfläche angezeigt.

The screenshot shows a web browser window titled "Solr Suchservice" with the URL "localhost:8082". The page is titled "TUBSearch beta" and features the heading "Suchmaschine. Mit Solr.". Below this, it says "Anzahl der gefundenen Ergebnisse: 135". A search bar contains the query "Student". To the right is a sidebar titled "Filter" with several categories and their counts: ITM (96), Service (49), News (47), Über uns (37), Studium und Lehre (32), Fachgebiet IuK-Management (29), and Lehrveranstaltungen (26). The main content area displays two search results:

- 1 Fakultät IV Elektrotechnik und Informatik: Kontakt**  
http://www.ise.tu-berlin.de/servicemenu/kontakt/  
student looking for study opportunities at the TU Berlin, please consult the following pages ... : International Admissions, prerequisites for a master's degree program, enrollment as a doctoral student or e ... Wichtiger Hinweis für Studienbewerber/innen: Wenn Sie sich für ein Studium an der TU Berlin ...  
Datum der letzten Indizierung/Änderung: Sat Jan 31 2015 06:05:08 GMT+0100 (Mitteleuropäische Zeit)
- 2 Fakultät VII Wirtschaft & Management: Externe Ausschreibung: Software und Student Developer bei Cringle**  
http://www.lkm.tu-berlin.de/menue/service/news/externe\_ausschreibung\_software\_und\_student\_developer\_be...  
Externe Ausschreibung: Software und **Student** Developer bei Cringle Zur Unterstützung ... der Softwareentwicklung sucht Cringle Software und **Student** Developer. Software Developer (PDF, 201,3 KB) **Student** Developer ... (PDF, 124,5 KB) Kontakt, Index und weiterer Service Kontakt, Inhaltsverzeichnis und weitere Service ...

Abbildung 3-22: Anzeige der Suchergebnisse

- **Status Code: 204 No Content**

Es wurde kein Ergebnis gefunden, die die Antwort des Servers enthält dementsprechend keinen Inhalt. Die Oberfläche weist darauf hin.

The screenshot shows a web browser window with the search term "Studentenwohnheim" entered in the search bar. The search results area contains the message "Schade! Es wurden keine Suchergebnisse gefunden."

Abbildung 3-23: Keine Suchergebnisse gefunden

- **Status Code: 500 Request failed.**

Es konnte keine Verbindung zum Server aufgebaut werden. Dies ist z.B. der Fall, wenn der SolrStack nachträglich beendet oder die Internetverbindung getrennt wurde.

The screenshot shows a web browser window with the search term "Studentenwohnheim" entered in the search bar. The search results area contains the message "Fehler! Verbindung zum Server konnte nicht hergestellt werden."

Abbildung 3-24: Suchanfrage wurde nicht beantwortet

### 3.3.5.2 FILTERN UND BLÄTTERN

Zum Filtern der Ergebnisse können nun die gefundenen Facetten auf der rechten Seite verwendet werden. Dafür können eine oder mehrere Facetten ausgewählt werden, wobei erneut eine Anfrage über die REST-API gestellt und die Oberfläche entsprechend aktualisiert wird:

Request URL: <http://localhost:8082/rest/request?q=Student&start=0&rows=10&fq=Projektleiter&hl=true>  
Request Method: GET

Anzahl der gefundenen Ergebnisse: 1

The screenshot shows a search interface with a search bar containing 'Student' and a 'Suchen' button. To the right is a 'Filter' sidebar with three checkboxes: 'ITM (1)', 'Projektleiter (1)' (which is checked), and 'Über uns (1)'. The main content area displays one search result: '1 Fakultät VII Wirtschaft & Management: Dr. Repschläger, Jonas' with a link to his profile. Below the result is a timestamp: 'Datum der letzten Indizierung/Änderung: Sat Jan 31 2015 06:06:06 GMT+0100 (Mitteleuropäische Zeit)'. The result is enclosed in a light blue box.

Abbildung 3-25: Suchergebnisse mit aktivierter Filterung

Außerdem können die Ergebnisse jeweils in Zehnerschritten durchgeblättert werden, indem auf die entsprechende Seitenzahl am Ende der Seite geklickt wird. Die aktuelle Seite wird dabei jeweils farblich hervorgehoben.

The screenshot shows a search results page with a timestamp at the top: 'Datum der letzten Indizierung/Änderung: Sat Jan 31 2015 06:05:06 GMT+0100 (Mitteleuropäische Zeit)'. Below it is a result for 'Fakultät VII Wirtschaft & Management: Chair of Information and Communication Management' with a link. The text describes the chair's offerings and welcome message. At the bottom of the result box is another timestamp: 'Datum der letzten Indizierung/Änderung: Sat Jan 31 2015 06:05:09 GMT+0100 (Mitteleuropäische Zeit)'. Below the results is a navigation bar with pages numbered 1 through 10, where page 1 is highlighted in blue. The entire page has a green decorative border.

Abbildung 3-26: Anzeige der Blätterfunktion

Der Klick auf 4 löst dabei beispielsweise die folgende Anfrage aus:

Request URL: <http://localhost:8082/rest/request?q=student&start=30&rows=10&fq=&hl=true>  
Request Method: GET

### 3.3.5.3 AUTOMATISCHE VERVOLLSTÄNDIGUNG („SUGGESTIONS“)

Während der Eingabe eines Suchwortes in das Eingabefeld wird ab dem zweiten Buchstaben und für jeden weiteren Buchstaben wiederum eine AJAX-Anfrage an die REST-Schnittstelle gesendet und das zurückgelieferte Ergebnis (JSON) in ein Auswahlmenü integriert, um eine komfortable Vorschlagsliste zu realisieren. Verwendet wurde hierzu die „autocomplete“-Komponente von jQuery UI.

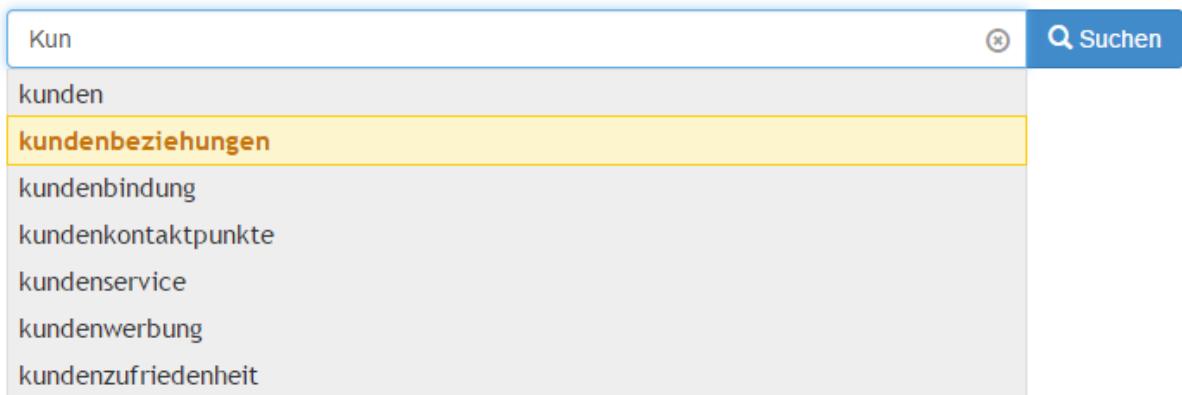


Abbildung 3-27: Anzeige der gefundenen Vorschläge

Wenn beispielsweise das Wort „Kunde“ eingegeben wird, sehen die einzelnen Anfragen an den Server wie folgt aus:

Name	Method	Status	Type
request?q=ku&rh=suggest	GET	200	application/json
request?q=kun&rh=suggest	GET	200	application/json
request?q=kund&rh=suggest	GET	200	application/json
request?q=kunde&rh=suggest	GET	200	application/json

Abbildung 3-28: aufeinanderfolgende AJAX-Anfragen bei Eingabe des Suchwortes „Kunde“

### 3.3.5.4 SONSTIGE FUNKTIONEN

Der Vollständigkeit halber seien noch folgende Funktionen erwähnt, die implementiert wurden:

- Bei einem Druck der Eingabetaste im geleerten Eingabefeld wird die Seite zurückgesetzt, d.h. die Ergebnisse oder sonstige Anzeigen werden von der Weboberfläche entfernt und die ursprüngliche Eingabemaske (siehe Abbildung 3-21: Eingabemaske) angezeigt.
- Dasselbe Verhalten kann mit einem Mausklick auf das „X“ im Eingabefeld erreicht werden, welches eingeblendet wird, sobald etwas eingetippt wurde.



**Abbildung 3-29: Schaltfläche zum Zurücksetzen der Oberfläche**

- Während auf eine Antwort des Servers gewartet wird, ist rechts oben eine animierte Ladeanzeige sichtbar.

Suchmaschine. Mit Solr.



**Abbildung 3-30: Ladeanzeige**

## 3.4 APACHE NUTCH

In diesem Abschnitt wird der Webcrawler Apache Nutch<sup>46</sup> beschrieben, welche Versionen es gibt und welche in diesem Projekt verwendet wird, aus welchen Bestandteilen Nutch besteht, wie er funktioniert und was bereits vorkonfiguriert wurde.

Nutch ist ein Webcrawler, der von Apache entwickelt wurde. Der Begriff Webcrawler bedeutet, dass Nutch automatisch das World Wide Web durchsucht und die Inhalte von Webseiten analysiert.

Nutch wird für die intelligente Suchmaschine benötigt um Inhalte von externen Webseiten, bei denen kein Zugriff auf das Content Management System(CMS) besteht, zu erhalten, parsen und die Inhalte in die Suchmaschine Solr zu transferieren. Nutch macht dies automatisiert mithilfe eines Skriptes und verfügt über eine integrierte Schnittstelle zu Solr, sodass Nutch als Webcrawler die geeignete Wahl ist.

### 3.4.1 VERSION

Von Nutch gibt es zwei verschiedene Serien, die sich in ihrer Grundarchitektur und Reife stark unterscheiden. Diese Serien sind die 1.x Serie und 2.x Serie und werden parallel bearbeitet und weiterentwickelt.

Die 1.x Serie ist ein reifer und „produktionsfertiger“ Webcrawler. Die aktuellste Version dieser Serie ist die Version 1.9, welche am 16. August 2014 von den Nutch-Entwicklern veröffentlicht wurde.

Die 2.x Serie baut auf der 1.x Serie auf, aber unterscheidet sich u.a. in der Speicherung der erhaltenen Daten. Hier werden die Informationen alle mithilfe des Backend Apache Gora<sup>47</sup> und einer Datenbank (z.B.: HBase<sup>48</sup>) in einer NoSQL-Tabelle gespeichert. Da die aktuellste Version 2.3 (22. Januar 2015) in einer frühen Entwicklungsphase ist, ist diese Serie noch nicht voll ausgereift und im produktiven Betrieb fehleranfälliger.

<sup>46</sup> <http://nutch.apache.org/> (letzter Zugriff: 14.3.2015)

<sup>47</sup> <http://gora.apache.org/> (letzter Zugriff: 05.03.2015)

<sup>48</sup> <http://hbase.apache.org/> (letzter Zugriff: 01.03.2015)

Nach dem Testen beider Serien und Abwägen der jeweiligen Vor- und Nachteile, fiel die Wahl letztendlich auf die 1.x Serie mit der aktuellsten Version 1.9.

### 3.4.2 BESTANDTEILE

bin/crawl.sh	Dies ist ein Bash-Skript, mit dem man den Crawlvorgang startet.
url/seed.txt	Eine Textdatei, in der die URLs enthalten sind, von denen der Crawlvorgang gestartet werden soll.
plugins/ crawl/ crawl/crawldb/	Dies ist ein Ordner, in dem alle Plug-Ins enthalten sind. Hier sind die Ergebnisse des Crawlvorganges enthalten. Enthält die Informationen (z.B.: fetchstatus) über jede bekannte Url.
crawl/linkdb/ crawl/segments/ conf/	Enthält eine Liste von URLs zu jeder URL (Webgraph). Enthält die gecrawlten Inhalte der URLs. Hier sind die Dateien zum konfigurieren des Crawlvorganges enthalten.
conf/regex-urlfilter.txt	In dieser Textdatei wird determiniert, welche URLs gecrawled werden.
conf/nutch-default.xml	Hier sind die Default-Eigenschaften des Crawlvorganges enthalten.
conf/nutch-site.xml	Die XML-Datei überschreibt die Eigenschaften der nutch-default.xml-Datei.
conf/solrindex- mapping.xml conf/extractor.xml	Definiert, welche Namen die Felder bei der Übertragung zu Solr bekommen. Hier sind die zu indizierenden Felder des Extractor Plug-Ins enthalten.
conf/parse-plugins.xml	Definiert, welche Plug-Ins bei welchen Mimetypes angewandt werden.
conf/schema.xml	Ein Beispiel für eine schema.xml, die in Solr eingefügt werden kann und mit Nutch kompatibel ist.

### 3.4.3 FUNKTIONSWEISE

Mithilfe des Bash-Skriptes crawl.sh aus dem bin-Ordner kann der Crawlvorgang gestartet werden. Dieses Skript führt folgende automatisierte Schritte aus:

1. Injecting (Einfügen der Start-URLs)
2. Generating (wählen der zu indizierenden Webseiten)
3. Fetching (Inhalt erhalten)
4. Parsing (umwandeln des Inhalt)
5. Update Crawldb
6. Link Inversion (Webgraph wird gebaut)
7. Merging mit Linkdb
8. Dedup der Crawldb (Duplikate entfernen)
9. SolrIndexing (Daten zu Solr transferieren)

## Recrawling

Das Bash-Skript crawl.sh kann ebenfalls zum Recrawling verwendet werden, indem man das Skript erneut ausführt. Bei erneuter Ausführung werden die bereits gecrawlten Webseiten nach dem Zeitpunkt des letzten Crawlvorganges geprüft. Ist dieser länger her als in der nutch-site.xml festgelegt (default:30 Tage), werden diese Dokumente erneut gecrawled. Ist dies nicht der Fall, werden diese übersprungen und anschließend werden die noch nicht indizierten Dokumente bearbeitet.

### 3.4.4 BISHERIGE KONFIGURATION

#### conf/nutch-site.xml

- fetcher.threads.per.queue=100 (Max. Anzahl Threads einer queue)
- fetcher.threads.fetch=100 (Anzahl der Fetcher-Threads)
- db.fetch.interval.default=2592000 (Zeit bis Refetching einer Seite=30 Tage)
- db.max.outlinks.per.page=-1 (Anzahl Outlinks unbegrenzt)
- fetcher.max.crawl.delay=10 (Wartezeit verringert)
- http.content.limit=65536 (Max. Größe von Dokumenten)
- db.ignore.internal.links=false (alle internen Links crawlen)
- db.ignore.external.links=false (alle externen Links crawlen)
- plugin.includes=extractor | parse-tika | ... (das Extractor Plug-In hinzufügt)

#### Extractor Plug-In

Da der Standard-Parser von Nutch die Inhalte der Webseiten nicht spezifisch genug filtert, sondern auch die Seitennavigationen und andere unerwünschte Inhalte mit indiziert, wurde ein zusätzliches Plug-In in Nutch eingebaut. Dieses ist das Extractor Plug-In und ist ein „Custom Search Tool<sup>49</sup>“ mit dem man bestimmte Ausschnitte von Html-Seiten mithilfe von css-Selektoren und xpath-Ausdrücken extrahieren und somit neue Felder definieren kann. Das Extractor Plug-In wurde von der „BayanGroup“ entwickelt und erweitert den Standard-Parser von Nutch. Aufgrund der Erweiterung lassen sich nur neue Felder definieren und keine bereits von Standard-Parser von Nutch verändern.

#### conf/extractor.xml

Hier sind die vom Extractor Plug-In neudefinierten Felder beschrieben. In der Grundkonfiguration wurden drei verschiedene Felder definiert. Als erstes wurden die Felder „htmlContent“ und „htmlTitel“ erstellt, indem die Inhalte und Titel der Webseiten beschrieben sind. Anschließend wurde noch das Feld „cat“ definiert, indem die Facetten der jeweiligen URLs enthalten sind. Zu Bemerken ist jedoch, dass man die hier definierten Ausdrücke für jede neue Webseitenstruktur neu definieren muss.

---

<sup>49</sup> <https://github.com/BayanGroup/nutch-custom-search> (letzter Zugriff: 27.3.2015)

## **conf/solrindex-mapping.xml**

Hier wurde konfiguriert, dass die vom Extractor erstellten Felder „htmlContent“ und „htmlTitle“ vor der Übertragung zu Solr in „content“ und „title“ umbenannt werden. Außerdem wurden die vom Tika-Parser erstellten Felder „content“ und „title“ in „extraContent“ und „extraTitle“ umbenannt.

## **conf/parse-plugins.xml**

In dieser Datei wurde definiert, dass für die Mimetypes „text/html“ und „application/xhtml+xml“ das Extractor Plug-In angewandt werden soll.

### **3.4.5 WAS KANN NUTCH?**

Der Webcrawler Apache Nutch kann mithilfe des automatisierten Bash-Skriptes crawl.sh verschiedene Dokumente von Webseiten fetchen, parsen und in Solr indizieren. Die unterstützen Dokumenttypen sind: Html, MsExcel, MsPowerPoint, MsWord, OpenOffice, PDF, Text.

## **4 SCHLUSS**

Im folgenden Abschnitt werden eine Zusammenfassung des Projektergebnisses sowie ein Ausblick für Verbesserungen am Suchservice aufgeführt. Dabei wird vor allem näher auf die Möglichkeiten von statistischen Auswertungen und das Thema Sicherheit eingegangen.

### **4.1 ZUSAMMENFASSUNG DES PROJEKTERGEBNISSES**

Dem Ziel einen Suchservice zu implementieren konnte entsprochen werden. Dabei sind folgende Meilensteine erreicht worden.

Es wurde ein SolrStack realisiert, mit dem effizient und effektiv nach Inhalten aus dem Index gesucht werden kann. Um dies zu ermöglichen wurden, die Schema- und die Konfigurationsdatei angepasst. Zum einen wurde die Grundkonfiguration umgesetzt und zum anderen wurden Suggestion und Facetten ermöglicht. Der SolrStack fungiert als Suchmaschine im Suchservice und kann mittels der REST-API angesprochen werden und mit dieser Informationen austauschen.

Ein funktionsfähiges Cluster ist eingerichtet. Es ist möglich, mit wenigen Änderungen an den Konfigurationen, den Suchservice als verteiltes System laufen zu lassen. Dabei werden Suchanfragen und Indizierung vom Clustermanager ZooKeeper schnell und zuverlässig auf die einzelnen Instanzen aufgeteilt, um Loadbalancing und Fault-Tolerance zu gewährleisten.

Der Webservice funktioniert in der geplanten Art und Weise. Es sind alle gewünschten Operationen über die bereitgestellte REST-API verfügbar gemacht und dank der erstellten Weboberfläche ist es möglich, bestimmte Teile der REST-API schnell über diese Oberfläche zu benutzen und die Ergebnisse ansprechend dargestellt zu bekommen. Die Kommunikation mit dem SolrStack funktioniert mittels SolrJ ohne Probleme.

Der Webcrawler Apache Nutch kann mithilfe des automatisierten Bash-Skriptes crawl.sh verschiedene Dokumente von Webseiten fatchen, parsen und in Solr indizieren. Die unterstützen Dokumenttypen sind: Html, MsExcel, MsPowerPoint, MsWord, OpenOffice, PDF, Text.

## 4.2 AUSBLICK

Zusätzlich zu den, in diesem Dokument beschriebenen Funktionen und Eigenschaften des Webservices, sind natürlich Erweiterungen und Verbesserungen möglich. Nachfolgend soll ein kleiner Ausblick über die Möglichkeiten gegeben werden, den Webservice zu erweitern und weiter zu entwickeln. Dazu wurden zwei wichtige Themen herausgegriffen, nämlich die Auswertung von Suchanfragen und Nutzerverhalten (Statistik) sowie eine Möglichkeit, den Zugriff auf die REST-API nur für befugte Nutzer zu gestatten (Sicherheit).

### 4.2.1 STATISTIK

Im Folgenden Abschnitt wird zuerst erläutert, welchen Mehrwert statistische Auswertungen bieten um anschließend auf Umsetzungsmöglichkeiten einzugehen.

#### 4.2.1.1 WARUM STATISTIK?

Der eigentliche Nutzen einer Suchmaschine hängt entscheidend damit zusammen, ob die gewünschten Informationen für den suchenden Nutzer schnell und einfach aufzufinden sind. Ob diese Anforderungen erfüllt sind und wo gegebenenfalls Optimierungspotential in der Konfiguration besteht, lässt sich dabei oft erst im laufenden Betrieb, also während des Praxiseinsatzes und der damit einhergehenden Nutzung durch verschiedene Nutzer feststellen.

Dazu ist es sinnvoll, verschiedene Charakteristika, wie beispielsweise verwendete Suchwörter, Filtereinstellungen, Facetten und ggf. die Nutzung der automatischen Vervollständigung sowie die resultierende Anzahl der Suchtreffer zu erfassen. Daraus können verschiedene Statistiken und Übersichten erstellt werden, die Aufschluss über das Nutzerverhalten geben und Verbesserungspotentiale offenbaren können.

Vorstellbar wäre unter anderem eine Übersicht über die meistverwendeten Suchbegriffe, um diese zum Beispiel über die automatische Vervollständigung bevorzugt vorzuschlagen. Auch können solche Informationen für den Aufbau des Index genutzt werden, um bessere Suchtreffer zu garantieren.

Ein wichtiges Kriterium ist die Relevanz der Suchergebnisse. Hierfür wäre denkbar, auch die sogenannte Click-Through-Rate (CTR) zu erfassen, also wie oft ein Nutzer einen bestimmten Link angeklickt hat. Idealerweise würde dann für eine Suchanfrage jeweils festgehalten, welcher Link jeweils geklickt wurde und auf welcher Position dieser bei den Suchergebnissen stand. Eine hohe Platzierung der wirklich relevanten Informationen im Suchergebnis ist hierbei der entscheidende Faktor.

#### 4.2.1.2 MÖGLICHKEITEN DER UMSETZUNG

Kommerzielle Analyse-Tools für Webseiten, wie Google Analytics<sup>50</sup> oder Piwik<sup>51</sup>, die in die eigene Webseite integriert werden, bieten eine breite Palette an Funktionen zur Analyse und Auswertung von Zugriffen auf die eigene Webseite und Nutzeraktionen, welche auch grafisch anspruchsvoll aufbereitet werden. Interessant ist hier die Möglichkeit, auch über eine seiteninterne Suche verwendete Suchwörter und Filter (hier dann die Solr-Suche) zu analysieren und auch die Anzahl der Klicks usw. festzuhalten.

Möchte man detailliertere Statistiken auch über die Daten im Index selbst, oder allgemein eine den eigenen Bedürfnissen besser angepasste Lösung erhalten, könnte auch eine eigene Statistik-Klasse implementiert werden, welche die Speicherung und Analyse der gewünschten Informationen übernimmt und dabei zusätzlich Solr-interne Funktionen verwendet. Insbesondere für die Analyse des Index selbst bieten sich zum Beispiel die *Stats Component*<sup>52</sup> und der *LukeRequestHandler*<sup>53</sup> an, welche direkte Statistiken über die gespeicherten Felder im Index und deren Werte liefern können.

Die erste Variante ist hierbei klar zu bevorzugen, da sie auf vielfach getestete und ausgereifte Lösungen zurückgreift, am schnellsten zu den gewünschten Informationen führt, ohne zu viel Konfigurations- und Programmieraufwand zu verursachen und bereits sehr detaillierte, das Nutzerverhalten betreffende Informationen liefern kann.

#### 4.2.2 SICHERHEIT

In der momentanen Implementierung kann die REST-API von jedem beliebigen Nutzer ohne vorherige Authentifizierung genutzt werden. Dies ist beim Stellen von Suchanfragen und dem Herunterladen von Dokumenten zwar auch absolut erwünscht, jedoch bietet sich damit natürlich auch für jeden die Möglichkeit, Dokumente zu erstellen, zu bearbeiten und zu löschen oder sogar Dateien auf den Server zu laden. Dies stellt eine große potentielle Gefahrenquelle dar, der entsprechend begegnet werden sollte.

Da eine REST-Anfrage immer zustandslos ist, muss eine Authentifikation bei jeder Anfrage durchgeführt werden, der Anfragesteller muss also die entsprechenden Informationen bei jeder Anfrage mitsenden. Eine relativ einfache und sichere Möglichkeit, eine REST-API entsprechend wirksam abzusichern, wäre die Nutzung eines sogenannten Keyed-Hash Message Authentication Code (HMAC).

Bei dieser Methode werden die eigentliche REST-Anfrage und weitere Informationen, wie Passwort (als Hashwert, auch „private hash“), Zeitstempel usw. mittels einer Hashfunktion in einen Hashwert (dann HMAC genannt oder auch „public hash“) überführt und der eigentlichen Anfrage angehangen. Der Server kann mit den gesendeten Informationen (z.B.

<sup>50</sup> <http://support.google.com/analytics/answer/1012264?hl=de> (letzter Zugriff 20.03.2015)

<sup>51</sup> <http://piwik.org/docs/site-search/#enable-site-search-tracking-for-your-website> (letzter Zugriff 20.03.2015)

<sup>52</sup> <http://cwiki.apache.org/confluence/display/solr/The+Stats+Component> (letzter Zugriff am 10.03.2015)

<sup>53</sup> <https://wiki.apache.org/solr/LukeRequestHandler> (letzter Zugriff am 10.03.2015)

Username und HMAC sowie dem ihm bekannten „private hash“) ebenfalls einen Hashwert erzeugen und somit die Identität des Anfragestellers und die Integrität der Anfrage bestätigen, sofern die Werte identisch sind.

Durch weitere Maßnahmen, wie die zeitliche Begrenzung der Gültigkeit eines HMAC oder die einmalige Gültigkeit (eine Anfrage kann mit dem gesendeten Hashwert nicht mehrfach ausgeführt werden) kann die Sicherheit weiter erhöht werden<sup>54</sup>.

---

<sup>54</sup> <http://restcookbook.com/Basics/loggingin/> (letzter Zugriff am 10.03.2015)

## INHALTSVERZEICHNIS

<b>1</b>	<b>Handbuch .....</b>	<b>3</b>
<b>1.1</b>	<b>Suchservice starten .....</b>	<b>4</b>
<b>1.2</b>	<b>SolrStack als Cluster.....</b>	<b>4</b>
1.2.1	Konfiguration ZooKeeper.....	4
1.2.2	Ausführen von Solr .....	5
<b>1.3</b>	<b>Webservice.....</b>	<b>6</b>
1.3.1	Start-Parameter.....	6
1.3.2	REST-API .....	6
1.3.3	Weboberfläche.....	11
<b>1.4</b>	<b>Apache Nutch.....</b>	<b>12</b>
1.4.1	Crawlvorgang starten .....	12
1.4.2	Crawlvorgang abbrechen .....	12
1.4.3	Crawlinhalt löschen .....	12
1.4.4	Zielwebseiten ändern .....	12
1.4.5	Konfiguration-Allgemein.....	13
1.4.6	Name des Crawlers ändern .....	13
1.4.7	Anzahl Threads ändern.....	13
1.4.8	Anzahl Dokumente in einer Fetchlist.....	13
1.4.9	Verzögerung beim Fetchvorgang verändern .....	13
1.4.10	Zeit bis zum Refetching verändern .....	14
1.4.11	Anzahl der Outlinks einer Seite bestimmen.....	14
1.4.12	Felder umbenennen.....	14
1.4.13	Externe und Interne Links verwalten .....	14
1.4.14	Partitioning Modus ändern.....	14
1.4.15	Felder hinzufügen/verändern.....	15
1.4.16	Plug-Ins hinzufügen .....	16
1.4.17	Metatags indizieren .....	16
<b>2</b>	<b>Folien .....</b>	<b>17</b>

## 1 HANDBUCH

In diesem Handbuch wird erklärt wie der fertige Suchservice gestartet und genutzt werden kann.

Den fertigen Suchservice ist als Ordner mit der Bezeichnung „suchservice\_release“ herausgegeben. Dieser Ordner besteht aus den mehreren Unterordnern (Abbildung 1-1: Suchservice Release), die die einzelnen Bestandteile enthalten. Zunächst der Ordner „apache-nutch-1.9“, der den fertigen Crawler beinhaltet. Dieser wird für den ersten Start nicht unbedingt benötigt, da auch Inhalte über die REST-API indiziert werden können. Sollen viele Daten einer Webseite indiziert werden, ist der Crawler ein sinnvolles Hilfsmittel. Wie er eingesetzt werden kann, wird im Abschnitt 1.4 Apache Nutch erläutert. Im Ordner „solrStack“ befindet sich die eigentliche Suchmaschine Solr. Diese ist auf die gestellten Anforderungen angepasst und vollständig funktionsfähig. Die REST-API, über die der Suchservice angesprochen werden kann, befindet sich im Ordner „webservice“. In diesem Ordner ist auch eine Beispielwebseite, die mit der REST-API kommunizieren kann, enthalten. Weitere Informationen, wie die REST-API angesprochen werden kann, befinden sich im Abschnitt 0 Webservice. Der Ordner „zookeeper-3.4.6“ enthält die Clusterkonfiguration. Informationen dazu wie man den SolrStack entweder lokal oder als Cluster ausführt, findet man im Abschnitt 1.2 SolrStack als Cluster. Des Weiteren werden mit dem fertigen Suchservice dieses Handbuch und die Webanwendung „indexClient.html“ zum Indizieren von Dateien mitgeliefert.

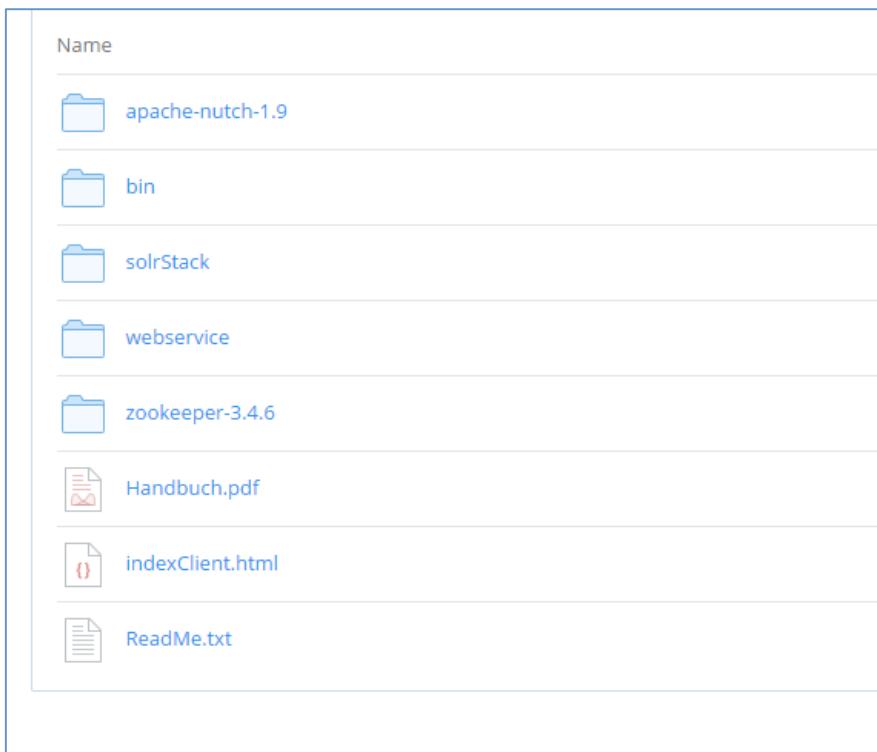


Abbildung 1-1: Suchservice Release

## 1.1 SUCHSERVICE STARTEN

Um den Suchservice zu starten müssen zwei Schritte ausgeführt werden:

1. Damit der Suchservice funktioniert muss zunächst der SolrStack gestartet werden. Dies ist über einen einfachen Befehl in der Kommandozeile möglich. Dafür muss zuerst in den Ordner SolrStack navigiert werden. Der Befehl lautet:  
`java -jar start.jar`
2. Anschließend muss der Webservice gestartet werden, damit der Suchservice über die REST-API angesprochen werden kann. Dafür muss die Jar Datei „webservice-0.0.1-SNAPSHOT-jar-with-dependencies.jar“ im Ordner Webservice ausgeführt werden.

Die weiteren Konfigurationsmöglichkeiten beim Starten werden im Folgenden beschrieben.

## 1.2 SOLRSTACK ALS CLUSTER

Um das Solr Cluster zu starten ist es notwendig, sofern das Softwarepaket nicht in der VM ausgeführt wird, die Konfigurationsdateien des ZooKeepers anzupassen. Zu finden sind diese im *conf*-Ordner. Jeder Server verwendet eine eigene *.cfg*-Datei.

### 1.2.1 KONFIGURATION ZOOKEEPER

```
1  #----Data Directory----#
2  dataDir=/zookeeperdata/Server1
3
4  #----Zookeeper Settings----#
5  clientPort=2181
6  tickTime=2000
7  initLimit=5
8  syncLimit=2
9
10 #----Servers running-----
11 server.1=vm1:2888:3888
12 server.2=vm2:2889:3889
13 server.3=vm3:2890:3890
```

***dataDir***: Speicherort für ID und Serverfiles

***clientPort***: Port zur Anbindung von Clients

***tickTime***: Zeit eines "Ticks" in Millisekunden

**initLimit**: Anzahl der maximalen "Ticks" für Initialisierung

**syncLimit**: Anzahl der "Ticks" die ein Server maximal zurückfallen darf

**server.ID**: IP und Port auf denen die Server kommunizieren, dabei sollte das *ID* durch einen Integer zwischen 1 und 255 gewählt werden. Die IPs verweisen dabei auf den jeweiligen Server.

Jede auszuführende ZooKeeper-Instanz sollte dabei seine eigene Konfigurationsdatei haben. Des Weiteren ist es nötig, in jedem *dataDir*-Ordner eine endungslose Datei mit dem Namen "*myid*" abzulegen, welche als einzigen Inhalt eine Zahl besitzt. Diese Zahl entspricht einer der Server.IDs. Danach werden die Instanzen mit Hilfe der Startskripte im *bin*-Ordner ausgeführt.

#### 1.2.1.1 LINUX

Beispielaufruf in der Konsole: ZkServer.sh start zoo1.cfg

#### 1.2.1.2 WINDOWS

Beispielaufruf in der Konsole: ZkServer.cmd start zoo1.cfg

### 1.2.2 AUSFÜHREN VON SOLR

(Anmerkung: Ist auf den ZooKeeper zu verweisen, so werden ALLE laufenden Server [IP:Clientport] mit Kommata getrennt angeführt, dabei werden keine Leerzeichen verwendet)

Sollten Veränderungen an Solr-Einstellungen vorgenommen werden, so ist es nötig zunächst die neue Konfiguration an den ZooKeeper zu übertragen.

Dafür verwendet wird das *zkCli*-Skript, welches sich im „scripts/cloud-scripts“-Ordner befindet:

```
sh zkcli.sh -cmd upconfig -zkhost <host:port> -confname <name for configset> -solrhome <solrhome> -confdir <path to directory with configset>
```

Danach kann die erste Solr-Instanz ausgeführt werden:

```
java -Dbootstrap_confdir=./solr/collection1/conf -Dcollection.configName=myconf -DzkHost=<host:port> -DnumShards=<number of shards> -jar start.jar
```

Im Anschluss werden weitere Instanzen gestartet. Dafür ist es notwendig, für jeden Shard, eine Kopie des *SolrStack*-Ordners zu erstellen, in welchem folgender Befehl ausgeführt wird:

```
java -Djetty.port=<Shard/Replica Port> -DzkHost=<ZooKeeper URL> -jar start.jar
```

## 1.3 WEBSERVICE

Um den Webservice zu starten, ist entweder die Main-Methode in der brain.Mastermind-Klasse auszuführen, oder die jar-Datei zu starten. Beides kann mit oder ohne Parameter geschehen.

### 1.3.1 START-PARAMETER

Die Parameter können genutzt werden, um den Startmodus (im Folgenden nur „Modus“, entweder *Local* oder *Cluster*) und die URL für den zu verbindenden SolrStack bzw. den Zookeeper (im folgenden nur „URL“) festzulegen.

Dabei ist zwischen folgenden Fällen zu unterscheiden:

#### 1.3.1.1 START OHNE PARAMETER:

Der Modus und die URL werden aus der config.properties übernommen.  
Beispielaufruf in der Konsole: `java -jar webservice.jar`

#### 1.3.1.2 START MIT EINEM PARAMETER:

Der Parameter muss entweder *local* oder *cluster* sein. Legt fest, in welchem Modus gestartet wird. Die URL wird dann aus der config.properties eingelesen.  
Beispielaufruf in der Konsole: `java -jar webservice.jar local`

#### 1.3.1.3 START MIT ZWEI PARAMETERN:

Der erste Parameter muss entweder *local* oder *cluster* sein. Legt fest, in welchem Modus gestartet wird. Der zweite Parameter muss die URL sein.  
Beispielaufruf in der Konsole: `java -jar webservice.jar local http://localhost:8983/solr`

## 1.3.2 REST-API

Die Dokumentation der REST-API erfolgt nach dem Vorbild von:

<http://bocoup.com/weblog/documenting-your-api/>

### 1.3.2.1 ANFRAGE STELLEN

URL: rest/request

Method: GET

Required Parameters:

q [String]

- Der Suchbegriff. Kann auch aus mehreren, mit Leerzeichen getrennten Begriffen bestehen.
- Beispiel: ?q=Fakultät 4 Lehrstuhl ISE

Optional Parameters:

rh [String]

- Der RequestHandler. Standartwert: *select*
- Beispiel: ?rh=select

start [int]

- Das wievielte Suchergebnis das erste Zurückgegebene sein soll. Standartwert: 0
- Beispiel: ?start=0

rows [int]

- Wie viele Suchergebnisse (Anzahl der Einträge) zurückgegeben werden sollen. Standartwert: 99999
- Beispiel: ?rows=10

fq [String]

- Der FilterQuery String. Wird für Facetten genutzt. Die einzelnen gewählten Facetten müssen per Komma voneinander getrennt sein.
- Beispiel: ?fq=Lehrstuhl,ISE

hl [boolean]

- Ob Highlighting für die Antwort verwendet werden soll.

### Success Response:

Code: 200

Content: JSON Repräsentation von ResultObject. In diesem sind enthalten:

- numFound: Die Anzahl der Suchergebnisse
- start : Der *start* Wert (siehe oben)
- rows : Die *rows* Wert (siehe oben)
- result : Der Ergebnisse der Suche (JSON Repräsentation von SolrDocumentList)
- facetResult : Mögliche Facetten für diese Suchergebnisse (JSON Repräsentation von List<FacetResult>)
- highlighting : Snippets mit Highlighting für jedes gefundene Dokument (JSON Array)

Code: 204

- Anfrage war erfolgreich, jedoch gab es keine Suchergebnisse.

---

### 1.3.2.2 DOKUMENT HINZUFÜGEN (CREATE)

URL: rest/document/<id>

Method: POST

Content-Type: application/xml

Http Body:

```
<add>
  <doc>
    <field name="id">id1</field>
    <field name="content">Lorem Ipsum!</field>
    <field name="url">http://www.example.de/</field>
  </doc>
  <doc>
    <field name="id">id2</field>
    <field name="content">Dolor sit.</field>
    <field name="url">http://www.example.de/amet</field>
  </doc>
</add>
```

### Success Response:

Code: 201

- Dokument wurde erstellt, Rückgabe der location(s) im Header bzw. Body (bei mehreren erstellten Dokumenten)

### Error Response:

Code: 500

- Serverfehler

---

#### 1.3.2.3 DOKUMENT LESEN (READ)

URL: rest/document/<id>

Method: GET

### Success Response:

Code: 200

- Ausgabe des Dokument als JSON

```
{  
    "id": "id1",  
    "content": "Lorem Ipsum!",  
    "content autocomplete": "Lorem Ipsum!",  
    "url": "http://www.example.de/",  
    "_version_": 1494594113995014100  
}
```

### Error Response:

Code: 404

- Dokument existiert nicht

---

#### 1.3.2.4 DOKUMENT BEARBEITEN (UPDATE)

URL: rest/document/<id>

Method: PUT

### Required Parameters:

field [String]

- Das Feld, das bearbeitet werden soll.

modifier [String]

- set, add oder remove

values [String]

- Werte, jeweils durch Komma getrennt

### Success Response:

Code: 204

- Dokument wurde entfernt

## Error Response:

Code: 404

- Dokument existiert nicht

---

### 1.3.2.5 DOKUMENT LÖSCHEN (DELETE)

URL: rest/document/<id>

Method: DELETE

## Success Response:

Code: 204

- Dokument wurde gelöscht

## Error Response:

Code: 404

- Dokument existiert nicht

---

### 1.3.2.6 DATEI INDIZIEREN

URL: rest/file/index

Method: POST

## Optional Parameters:

replace [boolean]

- Gibt an, ob eine Datei ersetzt werden soll, wenn diese bereits indiziert wurde. Standartwert: *false*

location [String]

- Der Ort, an dem die Datei auffindbar sein soll (beispielsweise eine URL). Wird keine *location* angegeben, wird die Datei im Dateisystem gespeichert.

## Http Body:

Beinhaltet eine Datei. Gesandt mit multipart/form-data.

## Success Response:

Code: 200

Content: Log des Vorganges.

#### Error Response:

Code: 302

Content: Log des Vorganges

- Die Datei existiert bereits, aber *replace* war *false*. Es wurde keine Indexierung vorgenommen.

---

#### 1.3.2.7 DATEI DOWNLOADEN

URL: rest/file/download

Method: GET

#### Required Parameters:

id [String]

- ID der Datei, der Datei, die gedownloaded werden soll.

#### Success Response:

Code: 200

Content: Datei als application/octet-stream. Der Dateiname ist die ID.

Code: 204

- Es existiert keine Datei mit dieser ID im Dateisystem.

#### Error Response:

Code: 404

- Der Server war nicht erreichbar.

---

#### 1.3.3 WEBOBERFLÄCHE

Die Weboberfläche ist über die IP in Verbindung mit dem voreingestellten Port 8082 zu erreichen. Auf dem eigenen System erfolgt der Aufruf z.B. über localhost:8082/.

Sie stellt eine komfortabel zu benutzende Oberfläche für das Durchführen von Suchanfragen und deren Filterung nach Facetten zur Verfügung.

## 1.4 APACHE NUTCH

In diesem Abschnitt ist beschrieben, wie man den Webcrawler Nutch bedient und zur Laufzeit den Crawlvorgang konfigurieren kann.

### 1.4.1 CRAWLVORGANG STARTEN

Zum Starten des Crawlvorganges wird das Bash-Skript crawl aus dem bin Ordner verwendet.

Format: bin/crawl <seedDir> <crawlDir> <solrURL> <numberOfRounds>

z.B.: bin/crawl url/ crawl http://localhost:8983/solr/ 1

### 1.4.2 CRAWLVORGANG ABBRECHEN

Der Crawlvorgang kann ohne größere Probleme jederzeit abgebrochen werden. Hierbei können jedoch temporäre Dateien im Nutch-Ordner erstellt worden sein, die vor der nächsten Ausführung des Skriptes manuell gelöscht werden müssen, da es sonst zu Komplikationen führen kann.

### 1.4.3 CRAWLINHALT LÖSCHEN

Um die gecrawlten Inhalte zu löschen reicht es den crawl-Ordner zu entfernen.

Hinweis: Das Löschen des crawl-Ordners bewirkt nicht, dass die Indizes aus Solr gelöscht werden.

### 1.4.4 ZIELWEBSEITEN ÄNDERN

#### **url/seed.txt**

In dieser Textdatei sind diejenigen Links enthalten, von denen aus der Crawlvorgang startet. Bei mehreren URLs müssen diese in jeweils eine Zeile geschrieben werden, wodurch auch mehrere URLs injiziert werden können.

z.B.: http://www.tu-berlin.de/

#### **conf/regex-urlfilter.txt**

Hier werden die Regeln definiert um zu entscheiden, welche Links gecrawled werden. Zur Bestimmung wird hier die URL betrachtet und so wird anhand der Regeln, welche mithilfe von regulären Ausdrücken („regex“) definiert werden, entschieden, ob diese URL gecrawled wird. Damit eine URL akzeptiert wird müssen alle Regeln erfüllt sein.

Mithilfe der Regeln können entweder bestimmte Ausdrücke mittels eines „-“ verboten werden oder bestimmte Ausdrücke mittels „+“ zwingend erforderlich gemacht werden.

Beispielsweise verbietet „-[?\*!@=]“ die Zeichen „?\*!@=“ in der URL und die Regel „+^(http|https)://\*.tu-berlin.de/“ wird verwendet damit jede URL die Folge „tu-berlin.de“ enthalten muss. Sind mehrere "+"-Regeln vorhanden reicht es aus, dass die URL mit einer übereinstimmt.

---

## 1.4.5 KONFIGURATION-ALLGEMEIN

### **conf/nutch-site.xml**

Die nutch-default.xml befindet sich im conf/ Ordner von Nutch. Hier befinden sich über 150 verschiedene Eigenschaften, mit denen der Webcrawler konfiguriert werden kann. Aufgrund der großen Anzahl der Eigenschaften und da zu jeder Eigenschaft eine Beschreibung in der nutch-default.xml vorzufinden ist, werden hier zu Gunsten der Übersichtlichkeit nur die wichtigsten Eigenschaften und Anwendungsfälle beschrieben.

### **conf/nutch-site.xml**

Um diese Grundeigenschaften des Webcrawlers zu verändern, wird die nutch-site.xml verwendet, mit der die Eigenschaften aus der nutch-default.xml überschrieben werden.

---

## 1.4.6 NAME DES CRAWLERS ÄNDERN

### **conf/nutch-site.xml**

Hier muss der Wert der Property ***http.agent.name*** auf den neuen Name geändert werden.

---

## 1.4.7 ANZAHL THREADS ÄNDERN

### **conf/nutch-site.xml**

Hier kann die Property ***fetcher.threads.per.queue*** verändern, damit die maximale Anzahl der Threads die beim Fetching in der Warteschlange sein dürfen festgelegt wird. Außerdem kann die Property ***fetcher.threads.fetch*** verändert werden, um die Anzahl der Threads zu bestimmen, die zum Fetching der Inhalte von den Webseiten eingesetzt werden sollen.

Hinweis: Die genaue Anzahl der zu verwendenden Threads ist bei jedem System unterschiedlich. Eine zu große Anzahl kann die Geschwindigkeit erheblich senken.

---

## 1.4.8 ANZAHL DOKUMENTE IN EINER FETCHLIST

### **conf/nutch-site.xml**

Hierzu wird die Property ***generate.max.count*** benutzt.

---

## 1.4.9 VERZÖGERUNG BEIM FETCHVORGANG VERÄNDERN

### **conf/nutch-site.xml**

Hierzu kann man die Property ***fetcher.max.crawl.delay*** verändern.

---

#### 1.4.10 ZEIT BIS ZUM REFETCHING VERÄNDERN

##### **conf/nutch-site.xml**

Um die Zeit bis zum nächsten Re-fetching zu bestimmen, kann die Properties ***db.fetch.interval.default*** zum Bestimmen des Intervalls, wann die URL das nächste Mal gefetched wird, benutzt werden und ***db.fetch.interval.max*** zum Bestimmen des Intervalls wann die URL unabhängig von ihrem Status versucht wird gefetecht zu werden. Beide Angaben erfolgen in Sekunden.

---

#### 1.4.11 ANZAHL DER OUTLINKS EINER SEITE BESTIMMEN

##### **conf/nutch-site.xml**

Hierfür wird der Wert der Property ***db.max.outlinks.per.page*** verändert. Beim Wert -1 ist die Anzahl unbegrenzt.

---

#### 1.4.12 FELDER UMBENENNEN

##### **conf/solrindex-mapping.xml**

Wenn ein von Nutch indiziertes Feld umbenannt werden soll bevor es zu Solr übertragen wird, kann diese Datei verwendet werden. Hierzu werden der alte Name mit source und der neue Name mit dest definiert.

z.B.:

```
<field dest="title" source="htmlTitle"/>
```

Hinweis: Das Feld muss in Solr so benannt sein, wie der dest-Wert ist.

---

#### 1.4.13 EXTERNE UND INTERNE LINKS VERWALTEN

##### **conf/nutch-site.xml**

Zum Ignorieren von internen bzw. externen Links von einer Webseite werden die Werte von ***db.ignore.internal.links*** bzw. ***db.ignore.external.links*** auf true gesetzt. Andernfalls werden diese auf false gesetzt.

---

#### 1.4.14 PARTITIONING MODUS ÄNDERN

##### **conf/nutch-site.xml**

Dies kann erreicht werden, indem das Feld ***partition.url.mode*** geändert wird.

#### 1.4.15 FELDER HINZUFÜGEN/VERÄNDERN

##### extractor.xml

Das Extractor Plug-In erweitert den Standard Parser von Nutch um neue Felder und kann nur für HTML-Dokumente verwendet werden. Folglich können neue Felder nur bei HTML-Dokumenten hinzugefügt werden und auch nur Felder indiziert werden, die nicht schon vom Standard Parser erstellt wurden.

Um Felder hinzuzufügen müssen zunächst die Felder im fields-Tag definiert werden.

z.B.:

```
<fields>
    <field> "htmlContent" />
    <field> "cat" multi="true" />
</fields>
```

Anschließend wird definiert bei welcher URL die jeweiligen Felder benutzt werden. Hierzu werden mithilfe von regulären Ausdrücken (regex) die anzuwendenden URLs in Dokumenten definiert. Bei mehreren möglichen Dokumenten wird das erste passende Dokument verwendet.

z.B. :

```
<document url="^http://([a-zA-Z0-9]*\.)*tu-berlin.de/" engine="css">
```

In den Dokumenten stehen die Felder und deren Inhalte beschrieben. Mit `<extract-to field="fieldname">` wird definiert, dass ein bestimmtes Feld erstellt wird.

z.B.:

```
<extract-to field="htmlContent">
```

Anschließend muss noch den Inhalt für die jeweiligen Felder definiert werden. Dies geschieht zunächst mit einem Tag `text` um aufzuzeigen, dass es sich um ein Text handelt und anschließend wird der Inhalt mittels `<expr value=>` bestimmt. Da das Extractor-Plug-In die Felder anhand der HTML-Elemente bestimmt, muss hier mittels Css-Selektoren, wie in css-Dokumenten, bestimmt werden, aus welchen Elementen der Inhalt für die Felder kommt.

z.B.:

```
<text>
    <expr value="#main .csc-default" />
</text>
```

Hinweis: Für die genaue Verwendung von Css-Selektoren sind hier weiterführende Informationen zu finden:

<http://wiki.selfhtml.org/wiki/CSS/Selektoren>

Hinweis: Bei ungeklärten Fragen zum Plug-In sind weitere Informationen zu finden auf

<https://github.com/BayanGroup/nutch-custom-search>

---

#### 1.4.16 PLUG-INS HINZUFÜGEN

##### plugin-Ordner

In diesem Ordner befinden sich alle Plug-Ins, die in Nutch enthalten sind. Die Plug-Ins in diesem Ordner müssen die auszuführenden .jar-Dateien enthalten und eine plugin.xml, welche abhängig vom Anwendungsfall die jeweilige jar-Datei determiniert.

##### conf/nutch-site.xml

Damit ein Plug-In auch von Nutch verwendet wird, muss hier die **plugin.includes** Property bearbeitet werden.

Um beispielsweise das Plug-In „Neues Plug-In“ einzufügen, sieht die Property anschließend folgendermaßen aus:

```
<property>
    <name>plugin.includes</name>
    <value>Neues_Plug-In|protocol-http|urlfilter-regex|...</value>
    <description> ... </description>
</property>
```

##### conf/parse-plugin.xml

In dieser Datei ist beschrieben welches Plug-In die jeweiligen Mimetype der gecrawlten Dokumente bearbeitet. Damit folglich ein neues Plug-In benutzt wird, muss es hier für den gewünschten Anwendungsfall definiert sein.

z.B.:

```
<mimeType name="text/html">
    <plugin id="extractor" />
    <plugin id="parse-html" />
</mimeType>
```

---

#### 1.4.17 METATAGS INDIZIEREN

##### conf/nutch-site.xml

1. Hier müssen zunächst im Feld **plugin.includes** die Werte parse-metatags und index-metadata eingetragen werden (siehe Plug-Ins hinzufügen), sofern noch nicht geschehen.

2. Im Feld **metatags.names** werden die gewünschten Werte eingetragen.

z.B.:

```
<value>description,keywords</value>
```

Im Feld **index.parse.md** müssen die zu indizierenden Werte mit dem Präfix metatag eingetragen werden.

z.B.:

```
<value>metatag.description,metatag.keywords</value>
```

3. Sofern gewünscht, können nun die Felder in der conf/solrindex-mapping.xml umbenennen werden (siehe Felder umbenennen).
4. Anschließend müssen die Felder noch in der schema.xml von Solr definiert sein (siehe Felder hinzufügen/verändern).

## 2 FOLIEN

Im Laufe des Projektes wurden die Projektfortschritte in vier Präsentationen vorgeführt. Diese waren die Projektplanvorstellung am 12.11.2014, Zwischenpräsentation 1 am 19.12.2014, Zwischenpräsentation 2 am 04.02.2014 und die Endpräsentation am 06.03.2015. Nachfolgend befinden sich die Folien zu den Präsentationen:

## Einführung



### Erste Schritte

- 1. Treffen mit Dr. Annette Bobrik - Projektleiterin seitens der TU Berlin
- Kick-Off bei adesso
  - Vorstellung Robby Reinicke
  - Unternehmensbeschreibung adesso
  - Vorstellung des Projektvorhabens und erwünschtes Ergebnis aus der Sicht von adesso
- nächste Schritte für uns - die Studenten
  - Erstellung des Projektplans mit detaillierter Beschreibung des Vorhabens

## Arten der Prototypen



### Muss-Prototyp

→ Besteht aus zwei Unter-Prototypen

→ Die finale Version des Muss-Prototyps soll das können, was Muss 2 kann

## Arten der Prototypen



### Muss 1

- ➡ Durch Grabber versorger Solr Server, der über eine Website erreichbar ist und als verteiltes System (Cluster) läuft

#### Vorgehen:

- Recherche und Aufbau der ersten Bestandteile
  - Grabber (auf Basis der example Datei)
  - Website/REST-API (auf Basis der example Datei)
  - autarker Solr Server (inkl. vollständiger Grundkonfiguration)
  - Clustersetup (auf Basis der example Datei)
- Verbindung der einzelnen Bestandteile
- Test von Muss 1

Arten der Prototypen  
Seite 5

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Arten der Prototypen



### Muss 2

- ➡ Muss 1 + die Möglichkeit über die Website Facetten einzustellen
- ➡ Der Grabber ist in der Lage Solr so zu versorgen, dass Facetten unterstützt werden

#### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
  - Grabber (zum korrekten Indizieren)
  - Website/REST-API (zum Auswählen und Anzeigen von Facetten)
  - Solr-Server/Cluster (Überprüfung ob Änderungen nötig sind)
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Muss 2

Arten der Prototypen  
Seite 6

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Arten der Prototypen



### Kann-Prototyp

→ Unterstützt die Analyse von Suchergebnissen und Anzeige von Suggestions

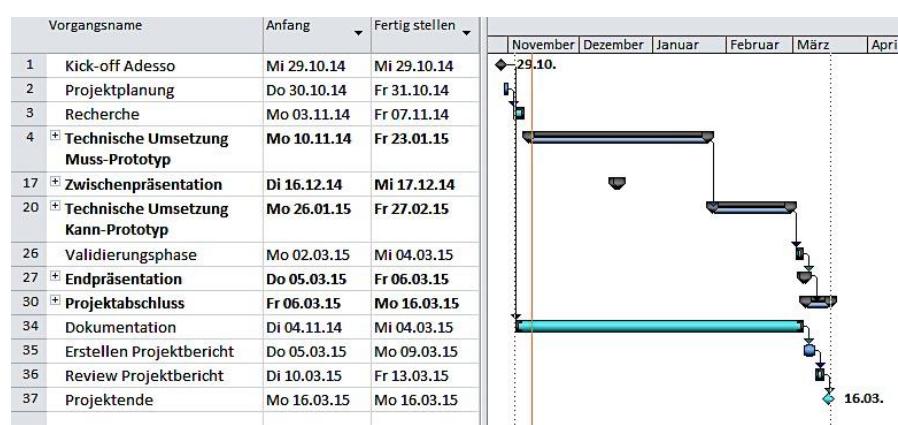
#### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
  - Website/REST-API (nach jedem Buchstaben Suchanfrage abschicken für Suggestions, Speicherung der Ergebnisse in Datenbank)
  - Grabber/Solr-Server/Cluster (Überprüfung ob Änderungen nötig sind)
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Kann-Prototyp

## Projektplan



### Übersicht Projektplanung



## Projektplan



### Termine

**Zwischenpräsentation:** Mi 17.12.2014

**Endpräsentation:** Fr 06.03.2014

### Präsentation Muss 2:

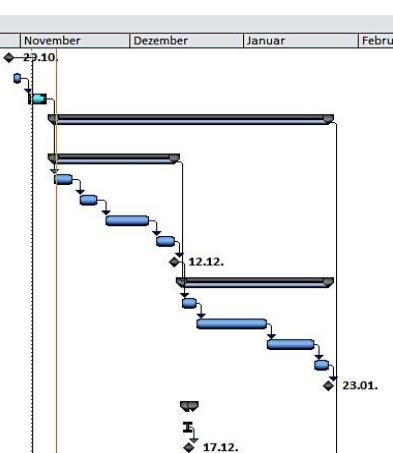
- Wenn dies erwünscht ist
- nach dem 23.01.2015

## Projektplan



### Projektschritte (1)

	Vorgangsname	Anfang	Fertig stellen
1	Kick-off Adesso	Mi 29.10.14	Mi 29.10.14
2	Projektplanung	Do 30.10.14	Fr 31.10.14
3	Recherche	Mo 03.11.14	Fr 07.11.14
4	■ Technische Umsetzung Muss-Prototyp	Mo 10.11.14	Fr 23.01.15
5	■ Prototyp1	Mo 10.11.14	Fr 12.12.14
6	Recherche	Mo 10.11.14	Fr 14.11.14
7	Konzeption	Mo 17.11.14	Fr 21.11.14
8	Umsetzung	Mo 24.11.14	Fr 05.12.14
9	Test	Mo 08.12.14	Fr 12.12.14
10	Release Prototyp 1	Fr 12.12.14	Fr 12.12.14
11	■ Prototyp2	Mo 15.12.14	Fr 23.01.15
12	Recherche	Mo 15.12.14	Do 18.12.14
13	Konzeption	Fr 19.12.14	Di 06.01.15
14	Umsetzung	Mo 07.01.15	Mo 19.01.15
15	Test	Di 20.01.15	Fr 23.01.15
16	Release Prototyp 2	Fr 23.01.15	Fr 23.01.15
17	■ Zwischenpräsentation	Di 16.12.14	Mi 17.12.14
18	Vorbereitung	Di 16.12.14	Di 16.12.14
19	Zwischenpräsentation	Mi 17.12.14	Mi 17.12.14



## Abschluss



### Weiteres Vorhaben

- Klärung Endpräsentation und Dokumentation
- Suchabfragen: Englisch und Deutsch?
- Kritik und Verbesserungen

Abschluss  
Seite 13

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



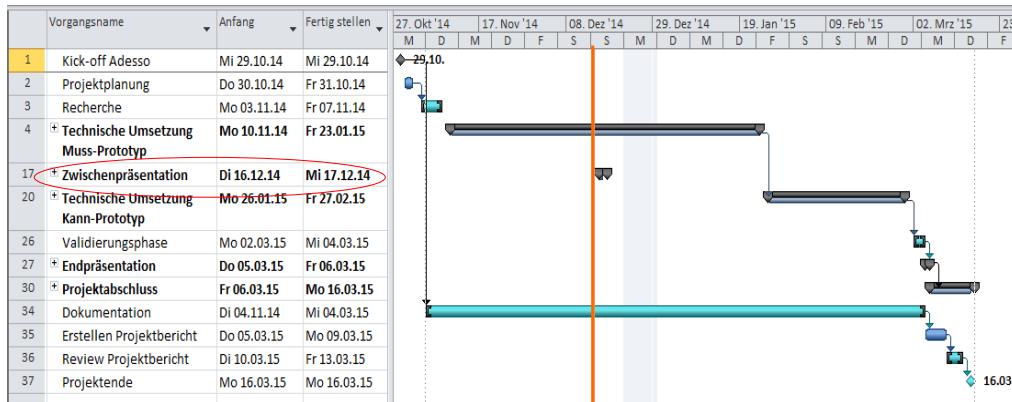
**Vielen Dank für Ihre**

**Aufmerksamkeit!**

Seite 14

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



Seite 3

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



### Rückblick

#### Anforderungen Muss 1

- Durch Grabber versorgter Solr Server, der über eine Website erreichbar ist und als verteiltes System (Cluster) läuft

#### Vorgehen:

- Recherche und Aufbau der ersten Bestandteile
  - Grabber (auf Basis der example Datei)
  - Website/Rest-API (auf Basis der example Datei)
  - autarker Solr Server (inkl. vollständiger Grundkonfiguration)
  - Clustersetup (auf Basis der example Datei)
- Verbindung der einzelnen Bestandteile
- Test von Muss 1

Seite 4

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



→ alle technischen Anforderungen von Muss 1 konnten umgesetzt werden

- Webseite (funktionsfähig mit unserem Solr-Server)
- Grundkonfiguration steht mit den einzelnen Komponenten
  - Umlaute
  - Stopwords
  - Caseing
  - Word Delimiter Filter Factory
- autarker Solr-Server
- Clustersetup (mit example Datei funktionsfähig)
- Grabber (funktionsfähig mit unserem Solr-Server)

Seite 5



## Umsetzung



# Website/Rest -API

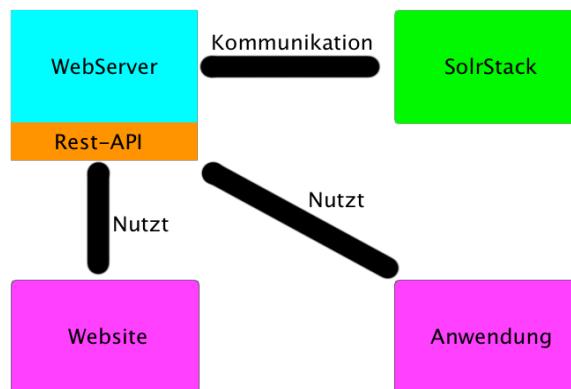
Seite 6



## Website / Rest-API



### Webseite



Seite 7

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Website / Rest-API



### Rest-API

Erstellt mithilfe von Jersey

```

@GET
@Produces({ "application/json", "application/xml" })
public ResultObject performRequest(
    @QueryParam("q") String q,
    @DefaultValue("0") @QueryParam("start") int start,
    @DefaultValue("99999") @QueryParam("rows") int rows) {
    logger.debug("Neue Anfrage: " + q);
    SolrDocumentList result = Mastermind.solr.performQuery(q, start, rows);
    if (result.getNumFound() == 0) {
        throw new WebApplicationException(Status.NO_CONTENT);
    }
    return new ResultObject(result);
}
  
```

Seite 8

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Website / Rest-API



## Rest-API

Seite 9

**ISE**ngineering  
Wirtschaftsinformatik –

## Umsetzung



# Grundkonfiguration

**ISE**Engineering  
Wirtschaftsinformatik –  
Management Science

## Grundkonfiguration



### Technische Details

- Anpassen von schema.xml an die deutsche Sprache
- Insbesondere durch Veränderung des FieldTypes "text\_general"

schema.xml in drei Teile eingeteilt

1. Definition der Felder
2. Definition der Feldtypen
3. Definition der allgemeinen Einstellungen

Seite 11



Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



```

468 <fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
469   <analyzer type="index">
470     <tokenizer class="solr.WhitespaceTokenizerFactory"/>
471     <filter class="solr.WordDelimiterFilterFactory"
472       generateWordParts="1"
473       generateNumberParts="1"
474       catenateWords="1"
475       catenateNumbers="1"
476       catenateAll="0"
477       splitOnCaseChange="1"
478       preserveOriginal="1"/>
479     <filter class="solr.LowerCaseFilterFactory"/>
480     <filter class="solr.SynonymFilterFactory"
481       synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
482     <filter class="solr.DictionaryCompoundWordTokenFilterFactory"
483       dictionary="dictionaryGerman.txt"
484       minWordSize="5"
485       minSubwordSize="3"
486       maxSubwordSize="30"
487       onlyLongestMatch="false"/>
488     <filter class="solr.StopFilterFactory"
489       words="stopwordsGerman.txt" ignoreCase="true"
490       enablePositionIncrements="true"/>
491     <filter class="solr.GermanNormalizationFilterFactory"/>
492     <filter class="solr.SnowballPorterFilterFactory"
493       language="German2" protected="protected.txt"/>
494     <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>

```

Seite 12

Information Systems Engineering

## Grundkonfiguration



### Feldtype:

- Menge von vordefinierten Feldtypen
- es gibt Felder, wie int oder date, die auf Standardtypen basieren
- Alle Feldtypen werden in der schema.xml innerhalb des XML-<types> definiert

<fieldType name="string" class="solr.StrField"/>

### Solr-Felder:

- einzelne Felder eines Index werden innerhalb des XML- Tags <fields> definiert
- Aufbau einer Definition grundlegend identisch
- Name und Typ einzige Pflicht-Properties
- alle weiteren können das Default- Verhalten vom Feldtypen überschreiben bzw. erweitern.

Seite 13



## Grundkonfiguration



### Allgemeine Einstellungen:

- in der schema.xml können zusätzliche Einstellungen gemacht werden
- haben Auswirkungen auf die Suche und die Auffindbarkeit von Dokumenten
- z.B.: Unique Key (eindeutige Identifier eines Solr- Dokuments)

### Der Analyse-Prozess:

- größter Einfluss auf den Index und auf die Auffindbarkeit von Dokumenten
- der Analyse-Prozess extrahiert aus dem Text die einzelnen Bestandteile und bereitet diese für die Suche auf.
- drei Phasen werden durchlaufen werden:
  - Charfilter
  - Tokenizer
  - Tokenfilter

Seite 14



## Grundkonfiguration



### Umlaute:

- Umlaute werden von Solr in der Beispielkonfiguration nicht erkannt
- German Normalization Filter Factory
  - berücksichtigt Besonderheiten der deutschen Sprache  
→ transformiert Umlaute (z.B. ä wird zu ae)

```
514
515 <filter class="solr.GermanNormalizationFilterFactory"/>
516
```

- Wird ein "Müller" gesucht, muss nicht dieses Schreibweise bekannt werden:
  - Müller → Mueller
- generell werden beide Schreibweisen gefunden:
  - März → Maerz

Seite 15

## Grundkonfiguration



### Stopwords:

- viele Wörter sollen bei der Suche keine Beachtung finden
- Stop Filter Factory
  - beinhaltet eine Textdatei mit zu ignorierenden Wörtern

```
525
526
527
528
529
530
531 <filter class="solr.StopFilterFactory"
532   words="stopwordsGerman.txt" ignoreCase="true"
533   enablePositionIncrements="true"/>
534
```

- Wörter wie aber, kein, ein, das werden bei der Suche nicht berücksichtigt
- wird nach " eine kleine Kirche im Wald" gesucht, sucht Solr nur nach "kleine" + "Kirche" + "Wald"
- wird nach "nur am Leben" gesucht, sucht Solr nur nach "Leben"

Seite 16

## Grundkonfiguration



### **Caseing:**

- bei Suchanfragen soll keine Rücksicht darauf genommen werden, ob der Suchbegriff klein oder groß geschrieben ist
  - Lower Case Filter Factory ermöglicht dies

```
501
502
503     <filter class="solrLowerCaseFilterFactory"/>
504
```

- Kirche ↔ kirche

## Grundkonfiguration



## Word Delimiter Filter Factory:

- für Worttrennung bei Zusammengesetzten Worte zuständig
  - auch Teilwörter, nicht nur das Gesamtwort wird gesucht
  - durch Parameter des Filters viele Anpassungen möglich (aktivieren oder deaktivieren)
    - z.B.: Trennung des Wortes bei
      - Wechsel von Groß- zu Kleinschreibung beachten
        - splitOnCaseChange
        - "WiFi"  $\longleftrightarrow$  "Wi", "Fi"
      - Bindestrichen
        - "Kreuz-Newsletter"  $\longleftrightarrow$  "Kreuz", "Newsletter"
      - Wechsel von Buchstaben und Zahlen
        - splitOnNumerics
        - "125Jährigen"  $\longleftrightarrow$  "125", "Jährigen"

## Grundkonfiguration



### Word Delimiter Filter Factory:

```

493      <filter class="solr.WordDelimiterFilterFactory"
494          generateWordParts="1"
495          generateNumberParts="1"
496          catenateWords="1"
497          catenateNumbers="1"
498          catenateAll="0"
499          splitOnCaseChange="1"
500          preserveOriginal="1"/>
501
...

```

- "1" aktiviert
- "0" deaktiviert

Seite 19

## Umsetzung



# Nutch

Seite 20

## Nutch



### Einleitung

- Nutch ist ein Webcrawler
- Dient zum Indexieren der Webseiten
- Open-Source
- Skalierbar
- Aktive Community

Seite 21



## Nutch



### Eigenschaften

- Multi-threaded fetcher
- Folgt Robot.txt
- „Freundliche“ Fetchregeln
- Konfigurierbar über Plugins
- Link-Ranking
- Url Filter

Seite 22



## Nutch



### Unterschiede Nutch 1x und 2x

#### Vorteile Nutch 1x:

- Stabiler
- Nicht so fehleranfällig
- Weniger zu installieren/konfigurieren
- Nicht abhängig von Backend Gora
- Transparenter

#### Vorteile Nutch 2x:

- Speicher in einer Tabelle
- Benutzt eigenen Webserver
- Ansprechbar über Rest-API

Seite 23



## Nutch



### Konfiguration - conf/regex-urlfilter.txt

- filtert welche URLs indexiert werden soll
- Verwendet verschiedene Regeln
- Mithilfe von Regulären Ausdrücken (regex)

#### Beispiele:

-[?\*!@=]  
+^(http|https)://.\*tu-berlin.de/

Seite 24



## Nutch



### Konfiguration – url/seed.txt

- Beschreibt Url von der aus gecrawled werden soll

Beispiele:

<http://www.tu-berlin.de/>

### Konfiguration – conf/schema.xml

- Beinhaltet Filter zum Indexieren
- Muss mit der schema.xml von Solr übereinstimmen

Seite 25



## Nutch



### Konfiguration – conf/nutch-site.xml

- Überschreibt conf/nutch-default.xml
- Definiert Eigenschaften des Crawls

Beispiele:

```
<property>
  <name>http.agent.name</name>
  <value>My First Crawl</value>
</property>

<property>
  <name>fetcher.threads.fetch</name>
  <value>100</value>
</property>
```

Seite 26



## Nutch



### Crawlen - Schritte

1. Injecting (Einfügen der Start-Uris)
2. Generating (wählen der zu indexierenden Webseiten)
3. Fetching (Inhalt erhalten)
4. Parsing (umwandeln des Inhalts )
5. Update CrawlDb
6. Link Inversion (Webgraph gebaut)
7. Merging with Linkdb
8. Dedup on CrawlDb (Duplikate entfernen)
9. SolrIndexing (Daten zu Solr übertragen)

Seite 27



## Nutch



### Crawlen - Skript

```
bin/crawl <seedDir> <crawlDir> <solrURL> <numberOfRounds>
```

Beispiel:

```
bin/crawl url/ crawl http://localhost:8983/solr/ 1
```

→ Kann auch zum Recrawling benutzt werden

### Crawlen - Ergebnisse

*crawlDb*: Informationen(Metadaten) über jede bekannte Url

*Linkdb*: Enthält Liste von Links zu jeder Url

*Segments*: Enthält gesammelten Inhalt überUrls

Seite 28



## Umsetzung



# Cluster

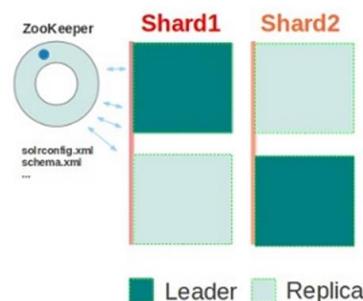
Seite 29

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Cluster - SolrCloud



- Zookeeper als Clustermanager
- Collection in bis zu vier Shards splitterbar
- Replicas für mehr Sicherheit erstellbar
- Automatisches Loadbalancing



Seite 30

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

# Cluster



Seite 31



# Fragen? Beispiele?

Seite 33





# offene Fragen?

Seite 35

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

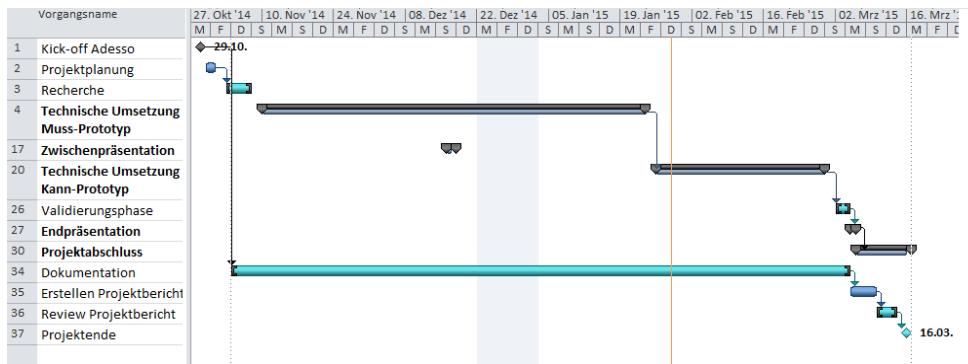


**Vielen Dank für Ihre  
Aufmerksamkeit!**

Seite 36

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



→ **Muss 1 + die Möglichkeit über die Website Facetten einzustellen**

Der Grabber ist in der Lage Solr mit Facetten zu unterstützen

### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
- Grabber (weiter Konfigurieren, in VM laufen lassen, Clusterfähig machen)
- Website/Rest-API (zum Auswählen und Anzeigen von Facetten)
- Solr-Server/Cluster (Überprüfung ob Änderungen nötig sind)
  - Zookeeper Konfiguration wird geändert
  - Zookeeper-Ensemble wird hinzugefügt
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Muss 2

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Status



→ alle technischen Anforderungen von Muss 2 konnten umgesetzt werden

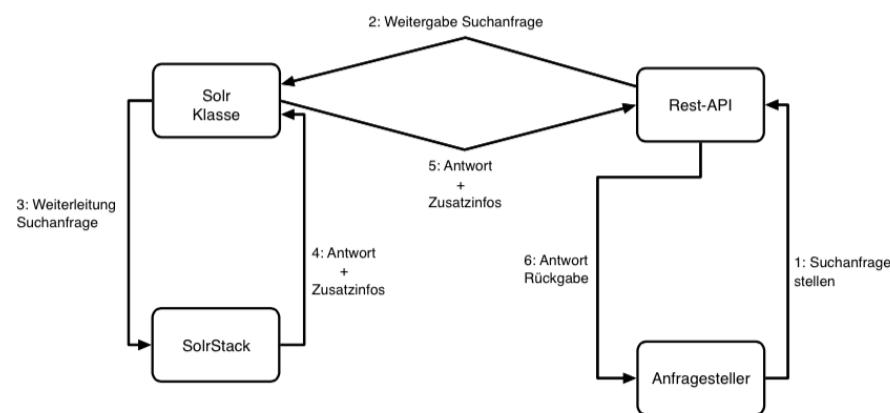
- Facettierung konnte implementiert werden:
  - Solr
  - Nutch
  - Webseite
- Suche läuft als Cluster in zwei virtuellen Maschinen
- Stemming umgesetzt
- Synonyme umgesetzt



## Architektur



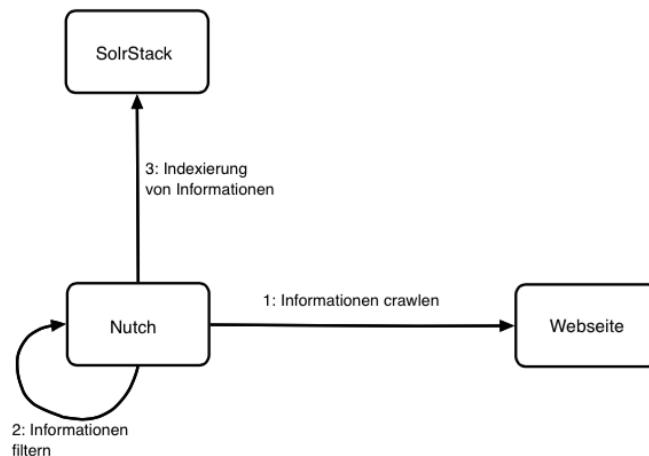
### Standardsuchanfrage



## Architektur



### Informationsversorgung

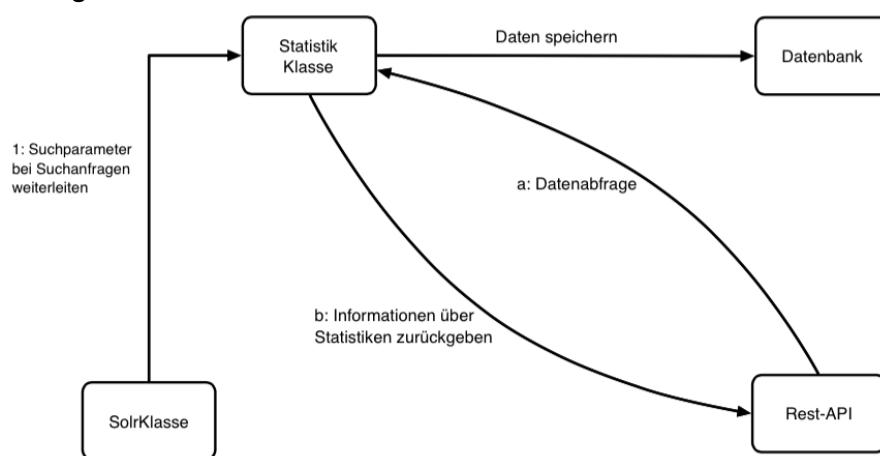


**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Architektur



### Anfragen zur Statistik



**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Konzept Rest-API



- Rest-API als Single Point of Contact (SPoC)

### Ablauf des Startens:

1. Starten des Webservices/der Rest-API
2. Benutzen der Rest-API um übrige Dienste zu starten/zu steuern
  - solrStack starten/stoppen
  - SolrKlasse mit solrStack verbinden
  - Nutch ausführen zum Crawlen
  - Suchanfragen stellen
  - Einstellungen vornehmen
  - Statistiken anzeigen



## Konzept Rest-API



### Allgemeine Funktionen

- solrStack starten/stoppen
- Solr Klasse (webservice) zum Verbinden mit solrStack konfigurieren
  - Parameter: IP-Adresse
- Verbindungsstatus von Solr Klasse anzeigen
- Verbindung der Solr Klasse (webservice) zum solrStack trennen
- Crawlvorgang Nutch starten
- Port für den webservice setzen
  - Parameter: neuer Port
- aktuellen Port webservice abfragen



## Konzept Rest-API



### Solr Klasse

- Suchanfrage stellen
  - Parameter:
    - start (ab dem wievieltem Ergebnis)
    - row (wie viele Ergebnisse)
    - q ("Query" -> Suchanfrage)
    - fq ("filterQuery" -> Suchergebnisse mit Facetten einschränken)
    - hl ("Highlighting" -> Wörter in Suchergebnis hervorheben)



## Konzept Rest-API



### Statistik Klasse

- Liefere Top Suchergebnisse
  - alle
  - 1te Buchstabe = ...
  - 1te&2te Buchstabe = ...
  - 1te&2te&3te Buchstabe = ...



## Facettierung

### Extractor Plugin

- Plugin/Extension für Nutch
- Erweitert den Standard-Parser von Nutch
- Von BayanGroup entwickelt
- Kompatibel mit Nutch 1.7-1.9
- „Custom Search Tool“
- Extrahiert bestimmte Ausschnitte von HTML Seiten
- Benutzt css-selectors oder xpath expressions



## Facettierung

### Plugin integrieren

nutch-site.xml

```
<property>
  <name>plugin.includes</name>
  <value>extractor|...</value>
</property>
```

parse-plugins.xml

```
<mimeType name="text/html">
  <plugin id="extractor" />
</mimeType>
```



## Facettierung

### **extractor.xml**

Felder definieren:

```
<fields>
    <field name="articleContent" />
    <field name="articleTitle" />
    <field name="lastUpdate" />
    <field name="cat" multi="true" />
</fields>
```



## Facettierung

### **extractor.xml**

Inhalt definieren:

```
<document url="^http://([a-z0-9]*\.)*tu-berlin.de/" engine="css">
    <extract-to field="articleContent">
        <text>
            <expr value="#main .csc-default" />
        </text>
    </extract-to>
</document>
```



## Facettierung



### extractor.xml

Kategorien definieren:

```
<document url="^http://([a-z0-9]*\.)*tu-berlin.de/" engine="css">
    <extract-to field="cat">
        <text>
            <expr value=".standort.current :not(em),.section em" />
        </text>
    </extract-to>
</document>
```

Dadurch erhält man bei der TU-Berlin Webseite Kategorien wie:

- Fachgebiet IuK-Management
- Studium und Lehre
- Team
- Forschung



## Facettierung



### schema.xml

Felder definieren:

```
<field name="cat" type="string" stored="true" indexed="true" multiValued="true"
/>

<field name="content" type="text_general" stored="true" indexed="true"/>
```



## Facettierung in Solr



- Bei einer facettierten Suche können Suchergebnisse anhand verschiedener Kategorien weiter gefiltert werden:
  - Auf einen Blick Übersicht der Suchergebnisse und Aufschlüsselung nach verschiedenen Kriterien
  - Keine Dead-ends: Anzahl der Ergebnisse zu einer bestimmten Facette werden angezeigt
  - Keine Reihenfolge zu beachten: Facetten können in beliebiger Reihenfolge ausgewählt werden



## Facettierung in Solr



- Datenelemente werden vom Indexer mit Schlagworten versehen  
→ Schlagworte sind die Grundlage für die facettierte Suche
- Facettierung wird beim Indexieren durchgeführt
- Die Facettierung Kommandos werden an die normale SolrQuery Anfrage angehängt
- Wird in der gleichen Query Antwort zurückgegeben



## Facettierung in Solr



3 Arten von Facettierung in Solr:

- Field facetting:
  - ruft entweder die Anzahl aller terms auf oder
  - nur die TopTreffer in jedem Feld
  - Das Feld muss indexiert werden
- Query facetting:
  - gibt die Anzahl der Dokumente der aktuellen Suche zurück, die zudem mit dem gegebenen Query übereinstimmen
- Date facetting:
  - gibt die Anzahl der Dokumente zurück, die in den aktuellen Datenbereich hineinfallen



## Facettierung in Solr



### Parameter:

facet	<ul style="list-style-type: none"> <li>• Wenn er auf "true" gesetzt ist, ermöglicht dies Facettierung</li> <li>• Ist er nicht aktiviert, verändern andere Einstellungen nichts</li> </ul>
facet.query	<ul style="list-style-type: none"> <li>• Spezifiziert einen Lucene query um einen facet count zu generieren</li> </ul>
facet.field	<ul style="list-style-type: none"> <li>• Name der ausgewählten Facette</li> </ul>

Beispiel Query:

<http://localhost:8983/solr/select?q=dell&wt=json&indent=true&facet=true>





**Suchmaschine.** Mit Solr.

Q. Suchen

Filter

Keine Facetten vorhanden

## Ablauf Anfrage Facettierung

Anfrage stellen

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



Anzahl der gefundenen Ergebnisse: 50

Filter

Keine Facetten vorhanden

<input checked="" type="checkbox"/>	<b>Fakultät IV Elektrotechnik und Informatik: Information Systems Engineering</b>
Datum der letzten Indizierung/Änderung: 28.01.2015	
Aktuelles Das FachgebietWir erforschen die software-technische Gestaltung und interdisziplinäre Bewertung von verteilten, auf Plattformen basierten IT-Systemen in anspruchsvollen, zukunftsweisenden Anwendungsbereichen - innovativ, technikorientiert, mit Praxisbezug und mit dem Anspruch auf internationales Spitzenniveau. Das Fachgebiet unter der Leitung von Prof. Dr.-Ing. Stefan Tai wurde im Juni 2014 gegründet und befindet sich noch im Aufbau. Kontakt, Index und weitere Service Kontakt, Inhaltsverzeichnis und weitere Service-Links	
<input checked="" type="checkbox"/>	<b>Fakultät IV Elektrotechnik und Informatik: Information Systems Engineering</b>
Datum der letzten Indizierung/Änderung: 28.01.2015	
Welcome to the research group Wirtschaftsinformatik - Information Systems Engineering (ISE), faculty IV, TU Berlin. News The research group Our topics in research and teaching are the design and assessment of distributed, platform-based IT systems in existing as well as emerging application domains. Using innovative, technical approaches backed by practical experience, we strive to shape the way tomorrow's state of the art enterprise IT applications will look like. The group, headed by Prof. Dr. Stefan Tai, was formed in June 2014 but builds on a long history as KIT's eOrganization research group. Kontakt, Index und weitere Service Kontakt, Inhaltsverzeichnis und weitere Service-Links	

## Ablauf Anfrage Facettierung

Antwort + alle passenden Facetten anzeigen

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Ablauf Anfrage Facettierung, Facetten filtern



<input type="checkbox"/> Institut Wirtschaftsinformatik und quantitative Methoden <b>50</b>	<input type="checkbox"/> Institut Wirtschaftsinformatik und quantitative Methoden <b>16</b>
<input type="checkbox"/> Team <b>18</b>	<input checked="" type="checkbox"/> Team <b>18</b>
<input type="checkbox"/> Lehre <b>10</b>	<input type="checkbox"/> Jörn Kuhlenkamp, M.Sc. <b>3</b>
<input type="checkbox"/> WS 2014/15 <b>10</b>	<input type="checkbox"/> Annette Bobrik, Dr.-Ing. <b>2</b>
<input type="checkbox"/> Teaching <b>4</b>	<input type="checkbox"/> Dominik Ernst, M.Sc. <b>2</b>
<input type="checkbox"/> Winter Term 2014/15 <b>4</b>	
<input type="checkbox"/> Enterprise Computing <b>3</b>	
<input type="checkbox"/> Information Systems Engineering <b>3</b>	

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



**Suchmaschine.** Mit Solr.

Anzahl der gefundenen Ergebnisse: 3

Tai

<input checked="" type="checkbox"/> Fakultät IV Elektrotechnik und Informatik: Jörn Kuhlenkamp, M.Sc. Datum der letzten Indizierung/Änderung: 28.01.2015 M.Sc. (Information Engineering and Management) Jörn Kuhlenkamp hat am Karlsruher Institut für Technologie (KIT) Informationstechnik studiert. Nach dem Abschluss seines Masterstudiums ist er als Wissenschaftlicher Mitarbeiter und Doktorand der Forschungsgruppe von Professor Tai im Bereich Cloud Computing beigetreten. Jörns Interessen liegen im Bereich von verteilten Softwaresystemen und Datenbanken. Projekte/Publikationen LINK Kontakt, Index und weiterer Service Kontakt, Inhaltsverzeichnis und weitere Service-Links	Filter
<input checked="" type="checkbox"/> Fakultät IV Elektrotechnik und Informatik: Jörn Kuhlenkamp, M.Sc. Datum der letzten Indizierung/Änderung: 28.01.2015 M.Sc. (Information Engineering and Management) Jörn Kuhlenkamp hat am Karlsruher Institut für Technologie (KIT) Informationstechnik studiert. Nach dem Abschluss seines Masterstudiums ist er als Wissenschaftlicher Mitarbeiter und Doktorand der Forschungsgruppe von Professor Tai im Bereich Cloud Computing beigetreten. Jörns Interessen liegen im Bereich von verteilten Softwaresystemen und Datenbanken. Projekte/Publikationen LINK Kontakt, Index und weiterer Service Kontakt, Inhaltsverzeichnis und weitere Service-Links	<input type="checkbox"/> Institut Wirtschaftsinformatik und quantitative Methoden <input checked="" type="checkbox"/> Jörn Kuhlenkamp, M.Sc. <input checked="" type="checkbox"/> Team

## Ablauf Anfrage Facettierung

Gefilterte Antwort + neue, passende Facetten anzeigen

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Synonyme



### SynonymFilterFactory

- Indiziert Synonyme für ein gegebenes Wort
- Textdatei enthält eine Synonymkette pro Zeile
- Funktioniert nicht für alle Synonyme  
→ Herausforderung mit Synonymen, die aus mehreren Wörtern bestehen!

Wird wie folgt eingebunden:

```
<filter class="solr.SynonymFilterFactory",
synonyms="synonymsGerman.txt" ignorecase="true,, expand="true"/>
```



## Stemming



- Verfahren, mit dem Wörter auf ihren Stamm zurückgeführt werden
  - z.B. die Deklination von: Hauses oder Häuser zu Haus
- Solr bietet verschiedene Algorithmen für die Unterstützung der deutschen Sprache an:
  - solr.SnowballPorterFilterFactory
  - HunspellStemFilterFactory



## Stemming



### SnowballPorterFilter

- Erlaubt das Stemming ohne Wörterbuch
- Eingesetzt mit dem GermanNormalizationFilter können auch bspw. bei der Suche nach „Früchte“ auch Ergebnisse mit dem Wort „Frucht“ gefunden werden

#### Vorteil:

- Große Suffixe werden entfernt, somit geringe Anzahl an Begriffen im Index

#### Nachteil:

- Suchergebnisse, die keinen Zusammenhang mit der Abfrage haben

#### → mögliche Lösung:

- StemmerOverrideFilterFactory: Textdatei override.txt notwendig
- KeywordMarkerFilterFactory: Wörter werden von Stemmer nicht berührt



## Stemming



### HunspellStemFilter

- Kein rein algorithmisches System
- Verwendet Datei mit grammatischen Regeln und Wörterbuch

#### Vorteil:

- Komplizierte grammatischen Regeln können angewendet werden
- Sucht nach mehreren gültigen Wortstämmen

#### Nachteil:

- Findet nur Wörter, die den Regeln entsprechen oder im Wörterbuch existieren
- Qualität von Hunspell hängt von Qualität des Wörterbuchs ab

→ HunspellStemFilterFactory

```
<filter class="solr.HunspellStemFilterFactory"
       dictionary="en_GB.dic" affix="en_GB.aff" ignoreCase="true" />
```



## Stemming



### Unsere Wahl: SnowballPorterFilter

- Die solr.SnowballPorterFilterFactory stellt zwei unterschiedliche Sprachattribute bereit: „Deutsch“ und „Deutsch 2“.
- „Deutsch 2“ modifizierte Version von „Deutsch“ zur Behandlung von Umlauten.
- Wurde wie folgt eingebunden:  
`<filter class="solr.SnowballPorterFilterFactory" language="German2" protected="protwords.txt"/>`
- Falls eine Regel 'zu aggressiv' für die Anwendung scheint, zusätzlich:  
`<filter class="solr.StemmerOverrideFilterFactory" dictionary="override.txt" ignoreCase="false"/>`
- Hindert den Stemmer daran die Wörter zu berühren:  
`<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt"/>`

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Demonstration



### Synonyme mit dem SynonymFilter:

- Da die übergebene Synonymliste eine Zeile mit den Synonymen "Schüler", "Student" beinhaltet, ergibt die Suche nach dem Wort "Schüler" Ergebnisse mit beiden Begriffen:
  - "Hilfe" &
  - "Unterstützung"

### Stemming mit dem SnowballPorterFilter:

- Suche nach "Lehrstuhl", "Lehrstuhls", "Lehrstühle" ergibt dieselbe Anzahl an Suchergebnissen, denn der Stemmer wandelt wie folgt um:
   
`"Lehrstuhl" -> "Lehrstuhl"`  
`"Lehrstuhls" -> "Lehrstuhl"`  
`"Lehrstühle" -> "Lehrstuhl"`

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## SolrCloud & ZooKeeper



### ZooKeeper Ensemble

- ZooKeeper Ensemble für erhöhte Failtolerance
- Zunächst drei ZooKeeper
- Anzahl sollte ungerade sein

### Virtuelle Maschinen

- Ubuntu VM (möglicherweise vorübergehend)
- zur Zeit zwei Virtuelle Maschinen
- beliebig erweiterbar - sollte als Testumgebung aber genügen



## Umsetzung



# Live Demo



## PDF und andere Formate



Mit dem Plug-In „TikaParser“ mehrere Formate möglich

- PDF
- Microsoft PowerPoint
- Microsoft Excel
- Microsoft Word
- Open Office
- HTML
- Text
- Weitere in Arbeit

Plug-In bereits in Nutch integriert

Kann zu Konflikten mit dem Extractor Plug-In führen



## Verschiedene Zeichensätze



### Solr kann alle Zeichen im UTF 8 Format einlesen

- die Dokumente müssen im UTF 8 Format in den Solr Server indexiert werden
- der Client muss UTF-8 URL entschlüsseln können um es an den Solr Server weiterleiten zu können
- der Server muss UTF 8 Strings verarbeiten können

### Nutch kann alle Unicode Formate einwandfrei indexieren



## Nächste Schritte



### Ausblick auf Kann 1

#### Ergebnis:

- Unterstützt die Analyse von Suchergebnissen und Suggestions
- Eventuell können die Suggestions mit Boosting verknüpft werden

#### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
  - Website/Rest-API (nach jedem Buchstaben Suchanfrage abschicken für Suggestions, Speicherung der Ergebnisse in Datenbank, eventuell gibt es ein einfaches Tool, das Speicherung und Anzeige erledigt)
  - Grabber/Solr-Server/Cluster (Überprüfung, ob Änderungen nötig sind)
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Kann-Prototyp



offene  
Fragen?





**Vielen Dank für Ihre  
Aufmerksamkeit!**

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



# PROJEKTÜBERSICHT

Seite 3

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Intelligente Suche



### Problemstellung

- Jeder Website Besucher erwartet heutzutage eine intelligente Suche á la Google
- Kommerzielle „mächtige“ Suchen sind oft zu teuer
- Solr bietet eine ebenso mächtige OpenSource Alternative, welche allerdings schon in der Grundkonfiguration nicht trivial zu nutzen ist
- Das Integrieren einer Suche in eine bestehende Website erfordert meist einen hohen Aufwand

### Ziel

- Einfaches Einbinden einer intelligenten Suche in einen Internetauftritt, in max. 2 Tagen

Quelle: Aufgabenbeschreibung Kick-Off  
Seite 4

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Arten der Prototypen



### Muss-Prototyp

- Besteht aus zwei Unter-Prototypen
- Die finale Version des Muss-Prototyp ist das Ergebnis von Muss 1 und Muss 2

### Kann-Prototyp

- Muss-Prototyp + Unterstützung von Suggestions und das manuelle Einfügen/Entfernen von Suchinhalten mit Facetten über die Rest-API

Arten der Prototypen  
Seite 7

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Projektplan



### Übersicht Projektplanung

Vorgangsnr.	Vorgangsnname	Anfang	Fertig stellen	
1	Kick-off Adesso	Mi 29.10.14	Mi 29.10.14	
2	Projektplanung	Do 30.10.14	Fr 31.10.14	
3	Recherche	Mo 03.11.14	Fr 07.11.14	
4	+ Technische Umsetzung Muss-Prototyp	Mo 10.11.14	Fr 23.01.15	
17	+ Zwischenpräsentation	Di 16.12.14	Mi 17.12.14	
20	+ Technische Umsetzung Kann-Prototyp	Mo 26.01.15	Fr 27.02.15	
26	Validierungsphase	Mo 02.03.15	Mi 04.03.15	
27	+ Endpräsentation	Do 05.03.15	Fr 06.03.15	
30	+ Projektabschluss	Fr 06.03.15	Mo 16.03.15	
34	Dokumentation	Di 04.11.14	Mi 04.03.15	
35	Erstellen Projektbericht	Do 05.03.15	Mo 09.03.15	
36	Review Projektbericht	Di 10.03.15	Fr 13.03.15	
37	Projektende	Mo 16.03.15	Mo 16.03.15	

Projektplan  
Seite 8

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



Projektübersicht  
**MUSS-PROTOTYP**

Seite 9

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

**Arten der Prototypen – Muss 1**



→ Durch Grabber versorgter Solr Server, der über eine Website erreichbar ist und als verteiltes System (Cluster) läuft

**Vorgehen:**

- Recherche und Aufbau der ersten Bestandteile
  - Grabber (auf Basis der example Datei)
  - Website/REST-API (auf Basis der example Datei)
  - autarker Solr Server (inkl. vollständiger Grundkonfiguration)
  - Clustersetup (auf Basis der example Datei)
- Verbindung der einzelnen Bestandteile
- Test von Muss 1

Arten der Prototypen  
Seite 10

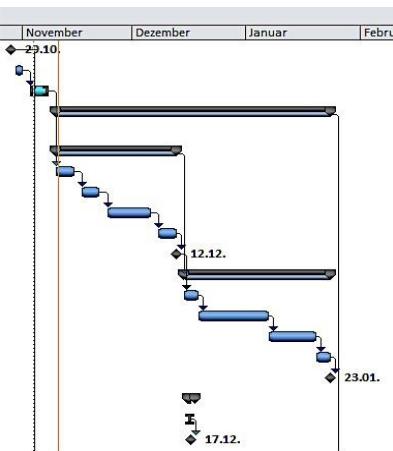
**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Projektplan – Muss-Prototyp



### Projektab schnitte (1)

	Vorgangsname	Anfang	Fertig stellen
1	Kick-off Adesso	Mo 29.10.14	Mo 29.10.14
2	Projektplanung	Do 30.10.14	Fr 31.10.14
3	Recherche	Mo 03.11.14	Fr 07.11.14
4	Technische Umsetzung Muss-Prototyp	Mo 10.11.14	Fr 23.01.15
5	Prototyp1	Mo 10.11.14	Fr 12.12.14
6	Recherche	Mo 10.11.14	Fr 14.11.14
7	Konzeption	Mo 17.11.14	Fr 21.11.14
8	Umsetzung	Mo 24.11.14	Fr 05.12.14
9	Test	Mo 08.12.14	Fr 12.12.14
10	Release Prototyp 1	Fr 12.12.14	Fr 12.12.14
11	Prototyp2	Mo 15.12.14	Fr 23.01.15
12	Recherche	Mo 15.12.14	Do 18.12.14
13	Konzeption	Fr 19.12.14	Di 06.01.15
14	Umsetzung	Mo 07.01.15	Mo 19.01.15
15	Test	Di 20.01.15	Fr 23.01.15
16	Release Prototyp 2	Fr 23.01.15	Fr 23.01.15
17	Zwischenpräsentation	Di 16.12.14	Mi 17.12.14
18	Vorbereitung	Di 16.12.14	Di 16.12.14
19	Zwischenpräsentation	Mi 17.12.14	Mi 17.12.14



Projektplan  
Seite 11

**ISE**ngineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Bestandteile

- Casing
- Stemming
- Umlaute
- Synonyme
- Stopwords

Arten der Prototypen  
Seite 12

**ISE**ngineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Technische Details

- Anpassen von schema.xml an die deutsche Sprache
- Insbesondere durch Veränderung des FieldTypes "text\_general"

Seite 13

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
    <analyzer type="index">
        <tokenizer class="solr.ClassicTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.SynonymFilterFactory"
            ...
            synonyms="synonymsGerman.txt" ignoreCase="true" expand="true"/>
        <filter class="solr.StopFilterFactory"
            words="stopwordsGerman.txt" ignoreCase="true"
            enablePositionIncrements="true"/>
        <filter class="solr.GermanNormalizationFilterFactory"/>
        <!--<filter class="solr.GermanStemFilterFactory"/><!--&gt;
        &lt;filter class="solr.SnowballPorterFilterFactory"
            language="German2" protected="protwords.txt"/&gt;
        &lt;filter class="solr.RemoveDuplicatesTokenFilterFactory"/&gt;
    &lt;/analyzer&gt;
    &lt;analyzer type="query"&gt;
        &lt;tokenizer class="solr.ClassicTokenizerFactory"/&gt;
        &lt;filter class="solr.WordDelimiterFilterFactory"
            ...
            protected="protwords.txt"/&gt;
        &lt;filter class="solr.LowerCaseFilterFactory"/&gt;
        &lt;filter class="solr.DictionaryCompoundWordTokenFilterFactory"
            ...
            protected="protwords.txt"/&gt;
        &lt;filter class="solr.GermanNormalizationFilterFactory"/&gt;
        &lt;filter class="solr.SnowballPorterFilterFactory"
            ...
            protected="protwords.txt"/&gt;
        &lt;filter class="solr.RemoveDuplicatesTokenFilterFactory"/&gt;
    &lt;/analyzer&gt;
&lt;/fieldType&gt;</pre>

```

Seite 14

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Caseing:

- bei Suchanfragen soll keine Rücksicht darauf genommen werden, ob der Suchbegriff klein oder groß geschrieben ist
- Lower Case Filter Factory ermöglicht dies

```
501
502
503
504 |   <filter class="solrLowerCaseFilterFactory"/>
      |   ...
      |
```

Seite 15

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Stemming

#### SnowballPorterFilter

- Erlaubt das Stemming ohne Wörterbuch
- Eingesetzt mit dem GermanNormalizationFilter können auch bspw. bei der Suche nach „Früchte“ auch Ergebnisse mit dem Wort „Frucht“ gefunden werden

#### Vorteil:

- Große Suffixe werden entfernt, somit geringe Anzahl an Begriffen im Index

#### Nachteil:

- Suchergebnisse, die keinen Zusammenhang mit der Abfrage haben

#### → mögliche Lösung:

- StemmerOverrideFilterFactory: Textdatei override.txt notwendig
- KeywordMarkerFilterFactory: Wörter werden von Stemmer nicht berührt

Seite 16

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Umlaute:

- Umlaute werden von Solr in der Beispielkonfiguration nicht erkannt
- German Normalization Filter Factory
  - berücksichtigt Besonderheiten der deutschen Sprache
    - transformiert Umlaute (z.B. ä wird zu ae)

```
514
515 | <filter class="solr.GermanNormalizationFilterFactory"/>
516
```

Seite 17

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Synonyme

#### SynonymFilterFactory:

- Indiziert Synonyme für ein gegebenes Wort
- Textdatei enthält eine Synonymkette pro Zeile
- Funktioniert nicht für alle Synonyme
  - Herausforderung mit Synonymen, die aus mehreren Wörtern bestehen!

```
<filter class="solr.SynonymFilterFactory"
       synonyms="synonymsGerman.txt" ignoreCase="true" expand="true"/>
```

Seite 18

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Grundkonfiguration



### Stopwords

- viele Wörter sollen bei der Suche keine Beachtung finden
- Stop Filter Factory
  - beinhaltet eine Textdatei mit zu ignorierenden Wörtern

```
529
530
531
532
533
534
```

```
<filter class="solr.StopFilterFactory"
       words="stopwordsGerman.txt" ignoreCase="true"
       enablePositionIncrements="true"/>
```

- Wörter wie „aber“, „kein“, „ein“, „das“ werden bei der Suche nicht berücksichtigt

Seite 19

## Arten der Prototypen – Muss 2



- ➔ Muss 1 + die Möglichkeit über die Rest-API Facetten einzustellen
- ➔ Der Grabber ist in der Lage Solr so zu versorgen, dass Facetten unterstützt werden

### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
  - Grabber (zum korrekten Indizieren)
  - Website/Rest-API (zum Auswählen und Anzeigen von Facetten)
  - Solr-Server/Cluster (Überprüfung ob Änderungen nötig sind)
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Muss 2

Arten der Prototypen  
Seite 20

## Facettierung in Solr



- Datenelemente werden vom Indexer mit Schlagworten versehen  
→ Schlagworte sind die Grundlage für die facettierte Suche
- Facettierung wird beim Indexieren durchgeführt
- Die Facettierungs Kommandos werden an die normale SolrQuery Anfrage angehängt
- Facetten werden in der gleichen Query Antwort zurückgegeben

Seite 21

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



## Projektübersicht **KANN-PROTOTYP**

Seite 22

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Arten der Prototypen – Kann-Prototyp



### Neuer Kann-Prototyp

→ Muss-Prototyp + Unterstützung von Suggestions und das manuelle Einfügen/Entfernen von Suchinhalten (Dokumenttypen, die von vom "<https://wiki.apache.org/solr/ExtractingRequestHandler>" unterstützt werden) mit Facetten über die Rest-API.

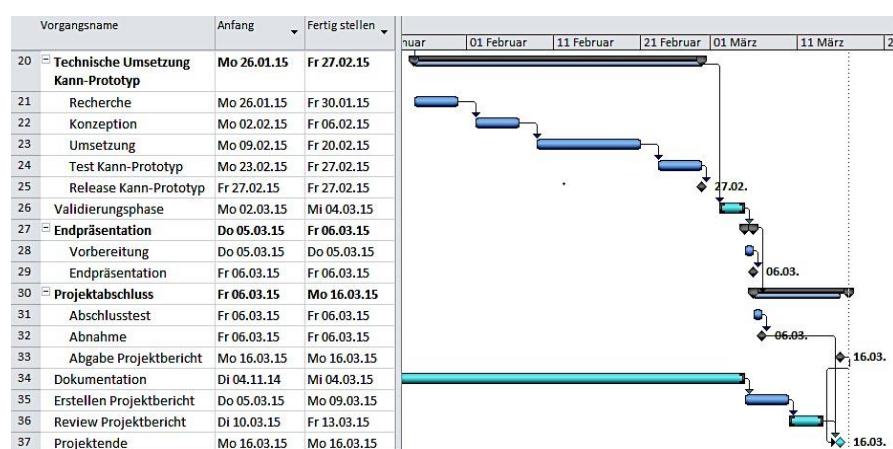
#### Vorgehen:

- Recherche und Veränderung der folgenden Bestandteile
  - Rest-API (neue Schnittstelle zur Annahme von Inhalten)
  - Grabber/Solr-Server/Cluster (Überprüfung ob Änderungen nötig sind)
- Integration der veränderten Bestandteile
- Validieren der Integration + Test Kann-Prototyp

## Projektplan - Kann-Prototyp



### Projektsabschnitte (2)



## Suggestion



### Autovervollständigung von Benutzereingaben

- durch einen neuen RequestHandler (/suggest) in der solrconfig.xml ermöglicht
- Wörterbuch das als Grundlage genutzt wird, wird aus den bereits indizierten Inhalten erstellt
- Ein neuer Feldtyp in der schema.xml, welcher die Informationen für Suggestions aufbereitet („text\_autocomplete“)

Seite 25

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Suggestion



```

<searchComponent name="suggest" class="solr.SpellCheckComponent">
  <lst name="spellchecker">
    <str name="name">suggest</str>
    <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
    <str name="lookupImpl">org.apache.solr.spelling.suggest.tst.TSTLookup</str>
    <str name="field">content_autocomplete</str>
    <str name="buildOnCommit">true</str>
  </lst>
</searchComponent>

<requestHandler name="/suggest" class="solr.SearchHandler" startup="lazy">
  <lst name="defaults">
    <str name="spellcheck">true</str>
    <str name="spellcheck.count">10</str>
    <str name="spellcheck.maxCollations">10</str>
    <str name="spellcheck.maxCollationTries">1000</str>
    <str name="spellcheck.collateExtended"></str>
  </lst>
  <arr name="components">
    <str>suggest</str>
    <str>query</str>
  </arr>
</requestHandler>

```

Seite 26

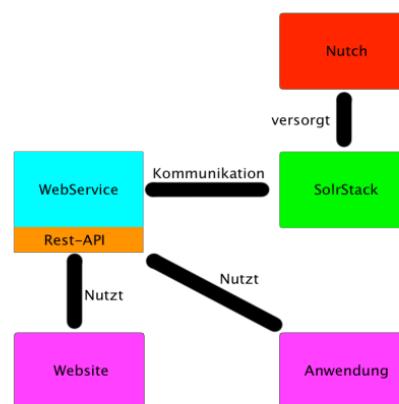
**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

# ARCHITEKTUR

Seite 27

## Architektur

### Grundaufbau des Suchservices

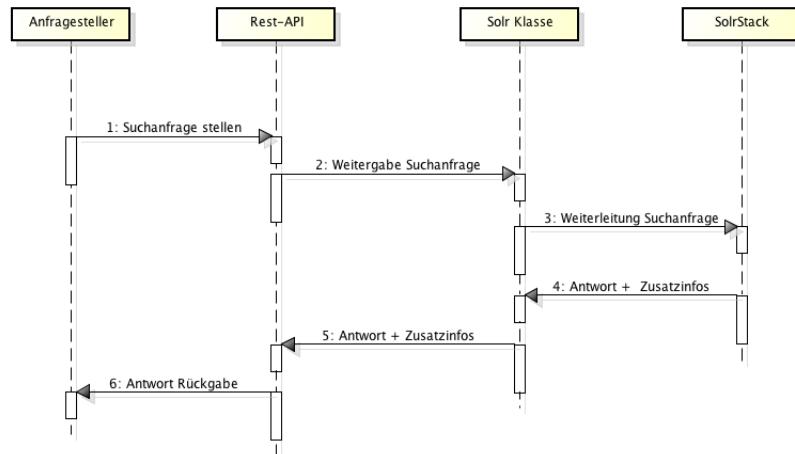


Seite 28

## Architektur



### Standardsuchanfrage



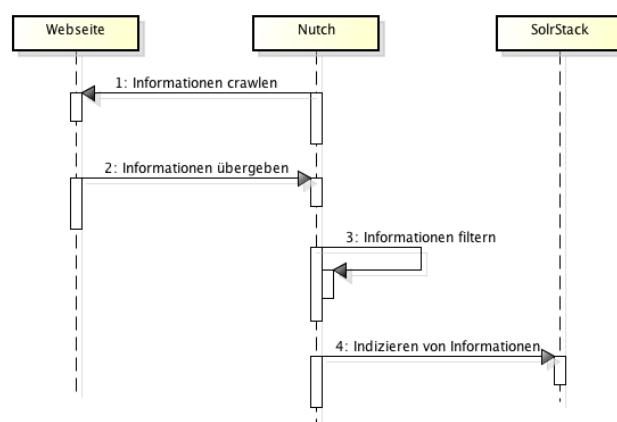
Arten der Prototypen  
Seite 29

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Architektur



### Informationsversorgung



Arten der Prototypen  
Seite 30

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



# REST-API

Seite 31

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Was kann unsere Rest-API?



- Suchanfragen stellen
- Suggestions anfordern
- Dokumente bearbeiten (Create, Read, Update, Delete)
- Dateien in Solr indizieren (pdf, rtf, txt,...)
- Dateien downloaden

Seite 32

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



Rest-API  
**SUCHANFRAGEN STELLEN**

Seite 33

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Suchanfragen stellen



URL: /rest/request  
Method: GET

### Required Parameters:

**q** Querystring, beliebiger Aufbau, Filterung möglich

### Optional Parameters:

**rh** RequestHandler (default: **select**)

**start** Startwert des ersten Suchergebnisses (default: **0**)

**rows** Beschränkung der gelieferten Suchergebnisse, beginnend bei **start** (default: **99999**)

**fq** FilterQuery-String, zur Filterung nach Facetten jeweils mit Komma getrennt (z.B. fq=Studium,IKM)

Seite 34

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Suchanfrage stellen



### Optional Parameters:

**hl** Angabe, ob Highlighting verwendet werden soll (default: **false**)

### Status Codes:

**200** Es wurden Suchergebnisse gefunden → JSON

- numFound: Die Anzahl der Suchergebnisse insgesamt
- start : Der *start*-Wert (siehe oben)
- rows : Der *rows*-Wert (siehe oben)
- result : Die Ergebnisse der Suche (JSON von SolrDocumentList)
- facetResult : alle Facetten und deren Häufigkeit (bezogen auf **alle** Ergebnisse)
- highlighting : Snippets mit Highlighting für jedes gefundene Dokument (JSON Array)

**204** Es wurden keine Suchergebnisse gefunden.

Seite 35

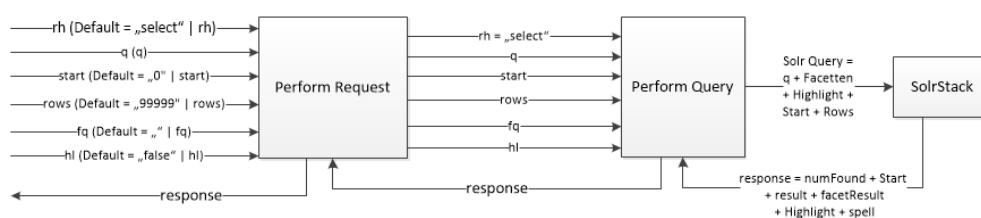


## Suchanfrage stellen



**URL:** /rest/request

**Method:** GET



Seite 36



## Suggestions anfragen

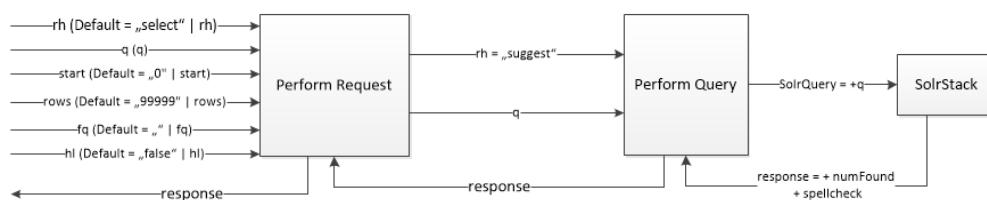


Benutzt die gleiche Schnittstelle wie  
Suchanfrage stellen!

Seite 37

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Suggestions anfragen



Seite 38

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



Rest-API  
**DOKUMENT BEARBEITEN**

Seite 39

**ISEEngineering**  
 Wirtschaftsinformatik –  
 Information Systems Engineering

### Dokument hinzufügen (CREATE)

URL: /rest/document/<**id**>  
 Method: POST  
 Content-Type: application/xml

#### Body:

```
<add>
  <doc>
    <field name="id">id1</field>
    <field name="content">Lorem Ipsum!</field>
    <field name="url">http://www.example.de/</field>
  </doc>
  <doc>
    <field name="id">id2</field>
    <field name="content">Dolor sit.</field>
    <field name="url">http://www.example.de/amet</field>
  </doc>
</add>
```



#### Status Codes:

**201** Dokument wurde indexiert / überschrieben  
 → Rückgabe der location(s)

```
Content-Length → 0
Location → http://localhost:8082/rest/document/id1
Server → Jetty(8.0.0.M1) [ "/rest/document/id1",
                           "/rest/document/id2" ]
```

**400** falsche Eingabe (XML)

Seite 40

**ISEEngineering**  
 Wirtschaftsinformatik –  
 Information Systems Engineering

## Dokument lesen (READ)



URL: /rest/document/<id>  
 Method: GET

### Status Codes:

**200** Dokument gefunden → Ausgabe als **JSON**

```
{
  "id": "id1",
  "content": "Lorem Ipsum!",
  "content autocomplete": "Lorem Ipsum!",
  "url": "http://www.example.de/",
  "_version_": 1494594113995014100
}
```

**404** Dokument mit <id> nicht gefunden

Seite 41



## Dokument bearbeiten (UPDATE)



URL: /rest/document/<id>  
 Method: PUT

### Required Parameters:

<b>field</b>	Feld im Dokument, das bearbeitet werden soll (z.B. cat)
<b>modifier</b>	- <b>set</b> (alle vorhandenen Werte ersetzen) - <b>add</b> (Werte hinzufügen) - <b>remove</b> (Werte entfernen)
<b>values</b>	Werte, jeweils durch Komma getrennt

### Status Codes:

**204** Änderungen erfolgreich ausgeführt

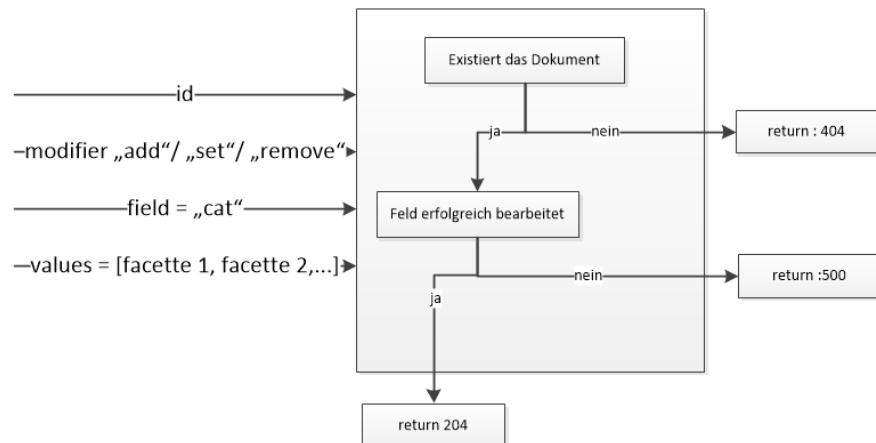
**404** Dokument mit <id> nicht gefunden

**400** falsche Eingabe, z.B. illegal modifier

Seite 42



## Dokument bearbeiten (UPDATE)



Seite 43

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Dokument löschen (DELETE)



URL: /rest/document/<**id**>  
 Method: DELETE

### Status Codes:

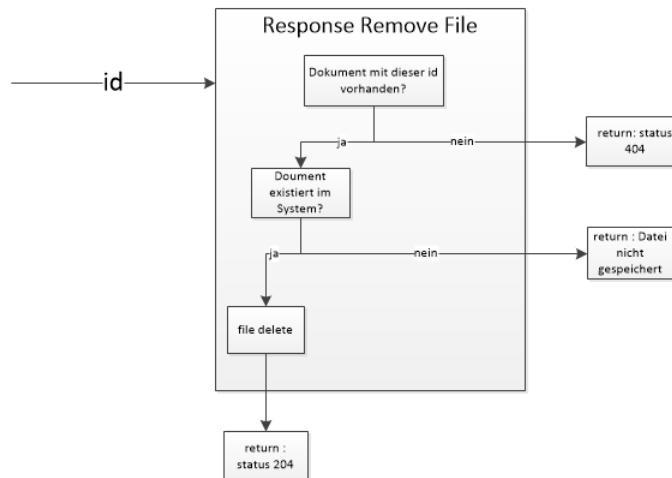
**204** Dokument wurde entfernt

**404** Dokument mit <id> nicht gefunden

Seite 44

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Dokument löschen (DELETE)



Seite 45

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



## Rest-API **INDIZIEREN**

Seite 46

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Datei in Solr indizieren



URL: /rest/file/index

Method: POST

Content-Type: multipart/form-data

Body: Die zu indizierende Datei

### Optional Parameters:

**replace** Datei ersetzen, falls bereits indiziert (default: **false**)

**location** Angabe über Zugriffsort der Datei (z.B. URL), ansonsten im Dateisystem gespeichert

### Status Codes:

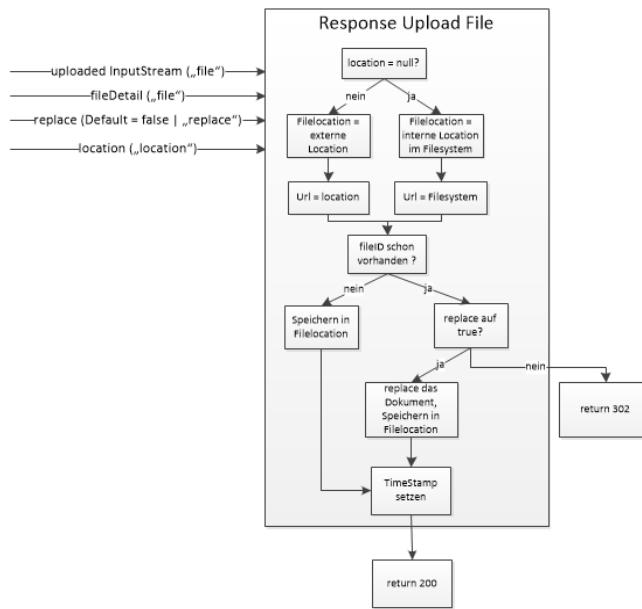
**200** Indizierung erfolgreich + Log des Vorganges

**302** Datei existiert bereits, replace=false, keine erneute Indizierung + Log des Vorganges

Seite 47

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Datei in Solr indizieren



Seite 48

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



Rest-API  
**DOWNLOADEN**

Seite 49

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

**Datei downloaden**



URL: /rest/file/download  
Method: GET

Required Parameters

**id** Id der Datei, die herunter geladen werden soll  
(entspricht dem Dateinamen)

Status Codes:

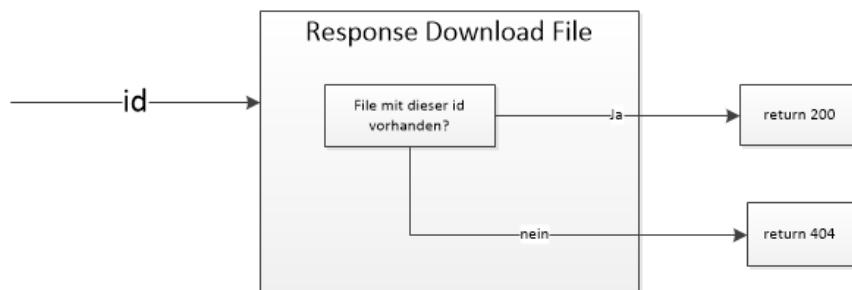
**200** Content: Datei als application/octet-stream

**404** Datei existiert nicht im Dateisystem

Seite 50

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Datei downloaden



Seite 51

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



## NUTCH

Seite 52

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Nutch - Allgemein



### Was ist Nutch?

- Webcrawler
- Dient zum Indizieren der Webseiten
- Open-Source
- Skalierbar
- Aktive Community
- viel dokumentiert
- Konfigurierbar über Plugins

### Wofür wird Nutch benötigt?

Inhalt von Webseite

- zu crawlern
- zu parsen
- zu indizieren
- schnell zu Solr transferieren

Seite 53



## Nutch - Version



### Welche Versionen gibt es?

- „1er“ Serie, 1.9 aktuellste Version (16.August 2014)
- „2er“ Serie, 2.3 aktuellste Version (22.Januar 2015)

### Was sind die wichtigsten Unterschiede?

- 2er Serie speichert Daten in externer Datenbank (z.B. HBase)
- 2er Serie verwendet eigenen Webserver

### Welche Version verwenden wir?

- 1.9

### Warum für 1.9 entschieden?

- Benutzerfreundlicher
- Stabiler
- Genügt für unsere Anforderungen

Seite 54



Nutch

# BESTANDTEILE &FUNKTIONSWEISE

Seite 55

## Nutch – Bestandteile (1)

### **bin/crawl.sh**

- Skript zum Indizieren der Webseiten

### **url/seed.txt**

- Enthält URLs der zu indizierenden Webseiten(Startpunkte)

### **conf/regex-urlfilter.txt**

- regelt welche Webseiten indiziert werden

### **conf/nutch-default.xml**

- Default Eigenschaften des Crawlvorganges

### **conf/nutch-site.xml**

- Überschreibt die Eigenschaften der nutch-default.xml

Seite 56

## Nutch – Bestandteile (2)



### **conf/solrindex-mapping.xml**

- Definiert welche Namen die Felder bei Übertragung zu Solr bekommen

### **conf/extractors.xml**

- Definiert die zu indizierenden Felder des Extractor Plug-Ins
- Zuweisung anhand von Html Code mittels css-Selektoren

### **plugins**

- Ordner, der alle Plug-Ins enthält

### **crawl**

- Ordner, der die gecrawlten Ergebnisse enthält

Seite 57



## Nutch – Funktionsweise (1)



### **Crawlvorgang**

1. Injecting (Einfügen der Start-Urls)
2. Generating (wählen der zu indizierenden Webseiten)
3. Fetching (Inhalt erhalten)
4. Parsing (umwandeln des Inhalts)
5. Update Crawldb
6. Link Inversion (Webgraph gebaut)
7. Merging with Linkdb
8. Dedup on Crawldb (Duplikate entfernen)
9. SolrIndexing (Daten zu Solr übertragen)

Seite 58



## Nutch – Funktionsweise (2)



### Crawl Skript

```
bin/crawl <seedDir> <crawlDir> <solrURL> <numberOfRounds>
```

Beispiel:

```
bin/crawl url/ crawl http://localhost:8983/solr/ 1
```

### Ergebnisse

Crawldb: Informationen(Metadaten) über jede bekannte Url

Linkdb: Enthält Liste von Links zu jeder Url

Segments: Enthält gesammelten Inhalt über Urls

Seite 59



Nutch

# GRUNDKONFIGURATION

Seite 60



## Nutch – Grundkonfiguration (1)



### **conf/nutch-site.xml**

- fetcher.threads.per.queue=100 (*Max. Anzahl Threads einer queue*)
- fetcher.threads.fetch=100 (*Anzahl der Fetcher-Threads*)
- db.fetch.interval.default=2592000 (*Zeit bis Refetching einer Seite=30 Tage*)
- db.max.outlinks.per.page=-1 (*Anzahl Outlinks unbegrenzt*)
- fetcher.max.crawl.delay=10 (*Wartezeit verringert*)
- http.content.limit=65536 (*Max. Größe von Dokumenten*)
- db.ignore.internal.links=false (*alle internen Links crawlen*)
- db.ignore.external.links=false (*alle externen Links crawlen*)
- plugin.includes=extractor | parse-tika | ...

Seite 61

## Nutch – Grundkonfiguration (2)



### **Extractor Plugin**

- Plugin/Extension für Nutch
- Erweitert den Standard-Parser von Nutch
- Definiert zusätzliche Felder
- Von BayanGroup entwickelt
- Kompatibel mit Nutch 1.7-1.9
- „Custom Search Tool“
- Extrahiert bestimmte Ausschnitte von HTML Seiten
- Benutzt css-Selektoren oder xpath expressions

Seite 62

## Nutch – Grundkonfiguration (3)



### Extractor - Felder defininieren

```
<fields>
  <field name="htmlContent" />
  <field name="htmlTitle" />
  <field name="cat" multi="true" />
</fields>
```

### Extractor - htmlContent definieren

```
<document url="^http://([a-z0-9]*\.)*tu-berlin.de/" engine="css">
  <extract-to field="htmlContent">
    <text>
      <expr value="#main .csc-default" />
    </text>
  </extract-to>
```

Seite 63

## Nutch – Grundkonfiguration (4)



### Extractor - Felder htmlTitle und cat definieren

```
<extract-to field="htmlTitle">
  <text>
    <expr value="title" />
  </text>
</extract-to>

<extract-to field="cat">
  <text>
    <expr value=".standort, #nav>ul>li>.current :not(em),>
      #nav-groups>ul>li>.current, .section em" />
  </text>
</extract-to>
</document>
```

Seite 64

## Nutch – Grundkonfiguration (5)



### **parse-plugins.xml**

```
<mimeType name="*">
  <plugin id="parse-tika" />
</mimeType>

<mimeType name="text/html">
  <plugin id="extractor" />
  <plugin id="parse-html" />
</mimeType>

<mimeType name="application/xhtml+xml">
  <plugin id="extractor" />
  <plugin id="parse-html" />
</mimeType>
```

Seite 65

## Nutch – Grundkonfiguration (6)



### **solrindex-mapping.xml**

```
<fields>
  <!-- Extractor-Plugin Felder -->
  <field dest="content" source="htmlContent"/>
  <field dest="title" source="htmlTitle"/>
  <field dest="cat" source="cat"/>
  <!-- Tika-Parser Felder -->
  <field dest="extraContent" source="content"/>
  <field dest="extraTitle" source="title"/>
  ...
</fields>
```

Wichtig: Felder müssen in schema.xml von Solr definiert sein

Seite 66

## Nutch - Grundkonfiguration(7)



### Was kann unser Nutch?

Indizieren von

- Html
- MsExcel
- MsPowerPoint
- MsWord
- OpenOffice
- PDF
- Text

Dateien

Erstellen von Feldern wie z.B.: Content, Autor, Titel, url

Übertragen der Daten nach Solr

Seite 67

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



Nutch

# KONFIGURATION ZUR LAUFZEIT

Seite 68

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering

## Nutch – Konfiguration (1)



### **url/seed.txt**

- Definiert Startpunkte des Crawlvorganges  
z.B.: <http://www.ise.tu-berlin.de>

### **conf/regexurl-filter.txt**

- filtert welcheUrls indexiert werden soll
- Mithilfe von Regulären Ausdrücken (regex)

z.B:

```
-[?*!@=]
+^(http|https)://.*tu-berlin.de/
```

Seite 69



## Nutch – Konfiguration (2)



### **conf/nutch-default.xml**

Enthält über 150 Einstellungen  
Unterteilt in u.a. :

- general Properties
- web db Properties
- generate Properties
- urlpartitioner Properties
- fetcher Properties
- indexingfilter plugin Properties
- plugin Properties
- parser Properties
- urlfilter plugin Properties
- solr index Properties

Seite 70



## Nutch – Konfiguration (3)



### **conf/nutch-site.xml**

- Überschreibt Properties des nutch-default.xml

### **conf/solrindex-mapping.xml**

- Zum Umbenennen der Feldnamen

### **conf/parse-plugins.xml**

- Definiert welche Plug-Ins wann angewendet werden

### **conf/extractors.xml**

- Definiert zusätzlich Felder für Html Dokumente

Seite 71

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



# CLUSTER

Seite 72

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering

## Cluster



- lief zunächst mit vorhandenem "SolrCloud"
  - durch Startskript im "bin" Ordner sehr schnell ausführbar
  - besitzt integrierten Cluster-Manager "ZooKeeper"
  - zuverlässiges Loadbalancing
  - Failtolerance
- Umstieg auf externen Apache ZooKeeper
  - erhöhte Failtolerance
  - erhöhte Skalierbarkeit
  - aber: erhöhter Konfigurationsaufwand bei großen Clustern
- Testumgebung für Cluster in Virtuellen Maschinen
  - Ubuntu (Server)

Seite 73

## Cluster



Seite 74

## Cluster - externe Zookeeper



- dataDir:  
Speicherort für ID und Serverfiles
- clientPort:  
Port zur Anbindung von Clients
- tickTime:  
Zeit eines "Ticks" in Millisekunden
- initLimit:  
Anzahl der maximalen "Ticks" für Initialisierung
- syncLimit:  
Anzahl der "Ticks" die ein Server maximal zurückfallen darf
- server.X:  
IP und Port auf denen die Server kommunizieren

```

1 #----Data Directory----#
2 dataDir=/zookeeperdata/Server1
3
4 #----Zookeeper Settings----#
5 clientPort=2181
6 tickTime=2000
7 initLimit=5
8 syncLimit=2
9
10 #----Servers running----#
11 server.1=vm1:2888:3888
12 server.2=vm2:2889:3889
13 server.3=vm3:2890:3890

```



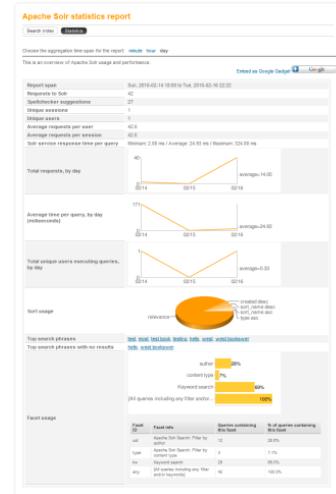
## Ausblick **STATISTIK**

## Ausblick



### Statistik - Tools

- kann von externen Tools übernommen werden
- Anwendung von z.B.:
  - Google Analytics
  - Piwik
  - etc.
- mit "Drupal" erstellte Seiten können Plug-In benutzen (siehe Abb.)
- sonstige Analysemethoden geben hauptsächlich Informationen über den Index



**ISE**Engineering

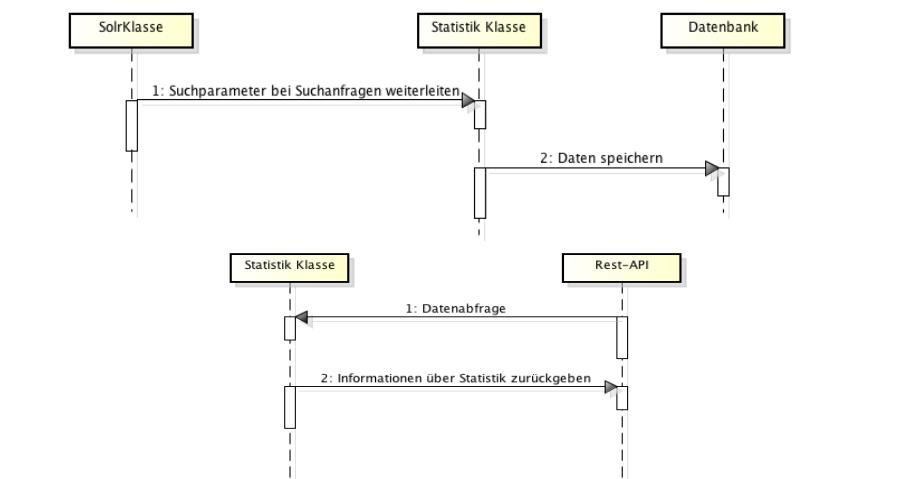
Wirtschaftsinformatik –  
Information Systems Engineering

Seite 77

## Ausblick



### Statistik - Architektur



Arten der Prototypen  
Seite 78

**ISE**Engineering  
Wirtschaftsinformatik –  
Information Systems Engineering



## LIVE DEMONSTRATION

Seite 79

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



offene  
Fragen?

Seite 80

**ISEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering



**Vielen Dank für Ihre  
Aufmerksamkeit!**

Seite 81

**ISEEngineering**  
Wirtschaftsinformatik –  
Information Systems Engineering