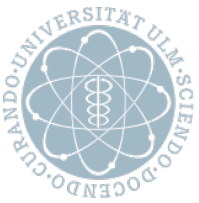

ulm university universität ulm

Fakultät für
Ingenieurwissenschaften und Informatik

Bachelorarbeit

DBIScholar - Konzeption und Entwicklung einer mobilen Anwendung sowie der dazugehörigen intelligenten Middleware mit RESTful Web Services

Vorgelegt von: Jonas Klan
jonas.klan@uni-ulm.de

Prüfer: Prof. Dr. Manfred Reichert

Betreuer: Marc Schickler

Abgabedatum: 16.12.2013

I. Kurzfassung

Applications (Apps) für mobile Geräte erfreuen sich nach wie vor immer mehr Beliebtheit und werden immer häufiger verwendet. Da mobile Geräte in den meisten Fällen nur über kleine Displays verfügen, per Touchscreen bedient werden und die Internetgeschwindigkeit von der Qualität des mobilen Netzwerkes abhängig ist, ist die Bedienung und die Verfügbarkeit von Internetdiensten erschwert. Durch übersichtliche und einfache Gestaltung und das Weglassen von Funktionen, erleichtern Apps den Zugriff auf diese Dienste. Das haben viele Firmen erkannt und bieten aus diesem Grund Apps für ihre Internetdienste an. Der App Store von Apple bietet zum Zeitpunkt dieser Arbeit beispielsweise mehr als 1 Million Apps zum Download an.
Derzeit sind einige Dienste, wie zum Beispiel die Literatur Suchmaschine des Unternehmens *Google Inc.*, *Google Scholar*, noch nicht von Google selbst realisiert. Um diese Lücke zu schliessen wird im Rahmen dieser Arbeit eine lauffähige iOS App für Google Scholar konzipiert und implementiert. Neben iOS existieren unter anderem noch die mobilen Betriebssystem Android und Windows, deshalb wurde in dieser Arbeit ein RESTful Web Service implementiert, der als eine Art Vermittler agiert. Dieser kann Anfragen von allen Geräten und Betriebssystemen entgegennehmen und die gefundenen Informationen auf Google Scholar per JavaScript Object Notation (JSON) zur Verfügung stellen. Somit ist es möglich, den Web Service für die Entwicklung anderer Software-Plattformen zu verwenden.

II. Inhaltsverzeichnis

III. Abkürzungsverzeichnis

API	Application Programming Interface
App	Application
H-Index	Hirsch Index
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internetprotokoll
JMS	Java Messaging Service
JSON	JavaScript Object Notation
RESTful	Representational State Transfer
SMTP	File Transfer Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
URI	Unique Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

1. Einleitung

Die ständige Weiterentwicklung und die vielfältigen Einsatzmöglichkeiten von Internetdiensten und mobilen Geräten führen zu einer zunehmenden Nutzung dieser Technologien. Dies hat unter anderem zur Folge, dass Literatur nicht mehr nur in Bibliotheken, sondern online über Suchmaschinen, wie *Google Scholar* gesucht wird.
Google Scholar stellt ein wichtiges Mittel in der wissenschaftlichen Literatur-Recherche dar und bietet mehrere Suchfunktionen an. Die gefundene Literatur wird durch ein Ranking-Verfahren sortiert, welches nur Google Scholar bekannt ist. Sicher ist dabei nur, dass häufig zitierte Werke weit oben in der Trefferliste stehen [8].
Es ist sehr hilfreich die gefundene Literatur nach ihrem literarischen Einfluss bewerten zu können. Dazu können die beiden bibliometrischen Maße Hirsch Index (H-Index) und G-Index verwendet werden. Google Scholar selbst stellt keine Application Programming Interface (API) bereit, die den Zugriff auf die gefundenen Informationen ermöglicht. Diese Aufgabe kann ein Parser übernehmen, der in der Lage ist, die Daten zur Verfügung zu stellen.
Google stellt keine App für Google Scholar zur Verfügung, die es ermöglicht, die Informationen auf einem mobilen Gerät einfach und übersichtlich darzustellen. Auf mobilen Geräten sind Apps benutzerfreundlicher als Webseiten, da sie sich auf kleinen Displays und der Bedienung per Touchscreens besser bedienen lassen.

1.1. Motivation

Die Entwicklung von Apps stellt sich als relativ aufwendig dar, weil es für jedes Betriebssystem eine eigene Entwicklungsumgebung und spezielle Schnittstellen gibt. Daher muss für jede Software-Plattform eine eigene, speziell auf das System zugeschnittene App implementiert werden. Aufgrund der Verbreitung vieler verschiedener Systeme auf mobilen Geräte ist es für die meisten Apps sehr wichtig, so viele Plattformen wie möglich zu erreichen. Eine App, die zu mehreren Plattformen kompatibel ist, erreicht mehr Anwender. Daher kommt eine Lösung, die Daten auf einem Endgerät zu parsen, wie es bereits in einer Diplomarbeit von Robecke 2011 an der Universität ausgeführt wurde nicht in Frage [13]. Aus diesem Grund wird im Rahmen dieser Arbeit ein RESTful Web Service konzipiert und implementiert, der die Daten von *Google Scholar* parst, die bibliometrischen Maße berechnet und plattformunabhängig per JSON zur Verfügung stellt. Dazu passend soll die zugehörige universelle native iOS App für *Google Scholar* entwickelt werden. Sie ist in der Lage, die verschiedene Suchfunktionen zur Verfügung zu stellen und die gefundenen Daten einfach und übersichtlich anzuzeigen. Aufgrund der RESTful-Architektur kann der Web Service auch für Apps auf anderen Betriebssystemen verwendet werden.

1.2. Aufbau der Arbeit

Die Ausarbeitung besteht aus sechs Teilen (siehe Abbildung 1). In Kapitel 2 werden die wichtigsten Grundlagen besprochen, um die Abläufe in den folgenden Kapiteln zu verstehen. In Kapitel 3 werden die funktionalen und nichtfunktionalen Anforderungen an den Web Service und die App beschrieben. Wie das gesamte System an sich funktioniert und wie die Teilaspekte der App und des Web Service implementiert sind, wird in Kapitel 4 erläutert. In Kapitel 5 wird schließlich tabellarisch aufgezeigt, ob die Anforderungen aus Kapitel 3 erfüllt, teilweise erfüllt oder nicht erfüllt wurden. Im abschließenden Kapitel 6 wird in einer Zusammenfassung ein Fazit formuliert und ein Ausblick gegeben.

Abbildung 1: Aufbau der Arbeit

2. Grundlagen

Für das Verständnis der nachfolgenden Kapitel, vor allem für die Implementierung des Systems, ist eine Einführung in die Grundlagen der einzelnen Komponenten erforderlich. Dabei ist *Google Scholar* die Datengrundlage für die Berechnung des *G-Index* und des *H-Index*. Der *Parser* ist zuständig für das Parsen der Daten, da Google Scholar keine API bereitstellt. Als Vermittler zwischen Anwender und Google Scholar fungiert der *Web Service*. Diese Komponenten werden in den nachfolgenden Unterkapiteln erläutert.

2.1. Google Scholar

Abbildung 2: Einfache Suche bei Google Scholar [8]

In Abbildung 2 ist die Webseite von Google Scholar, eine Suchmaschine für wissenschaftliche Literatur und Fachbücher, dargestellt. Mit Hilfe dieser Suchmaschine ist es möglich, auf die verschiedensten Quellen und Fachrichtungen zugreifen zu können. Diese sind zum Beispiel:

- Seminararbeiten
- Magister-, Diplom- sowie Doktorarbeiten
- Bücher
- Zusammenfassungen
- Artikel

Diese können aus akademischen Verlagen, Berufsverbänden, Magazinen für Vorabdrucke, Universitäten und anderen Bildungseinrichtungen stammen [8]. Um die Ergebnisse bewerten zu können, werden die verwandten Arbeiten, die Anzahl der Zitate, Autoren und Veröffentlichungen angezeigt. In Google Scholar gibt es zwei Möglichkeiten zu suchen.
Die *einfache Suche* mit nur einem Suchfeld (siehe Abbildung 2) und die *erweiterte Suche* mit der es möglich ist nach Artikeln mit den nachfolgenden Einschränkungen zu suchen.

- mit allen Wörtern
- mit der genauen Wortgruppe
- mit irgendeinem der Wörter
- ohne die Wörter
- die meine Wörter enthalten
- die von folgenden Autoren veröffentlicht wurden
- die hier veröffentlicht wurden
- die in folgendem Zeitraum geschrieben wurden

Google Scholar sortiert die Dokumente nach der gleichen Art und Weise wie es die Forscher selbst auch praktizieren [11]. Es wird der gesamte Text bewertet, also wo er publiziert wurde, von wem er geschrieben wurde und wie oft und wie häufig er in anderen wissenschaftlichen Arbeiten zitiert wurde [8].
Was Google Scholar nicht berücksichtigt, ist ein metrisches Maß zur Evaluierung der Publikationsleistung von Wissenschaftlerinnen und Wissenschaftlern. Als sehr nützlich haben sich dazu der H-Index und der G-Index erwiesen, welche in den nachfolgenden Kapiteln erläutert werden.

2.2. H-Index

Der H-Index ist ein Versuch den Einfluss und die Auswirkung von veröffentlichten Arbeiten von Wissenschaftlern oder Forschern zu messen. Er wurde 2005 von *J.E Hirsch* in einem Fachartikel eingeführt und ist folgendermaßen definiert: "an index to quantify an individual's scientific reseach output" [9].
Ein Wissenschaftler/Forscher hat einen H-Index von h, wenn seine N_p Publikationen mindestens h mal, die restlichen (N_p-h) Publikationen höchstens h-mal, zitiert wurden. Eine Möglichkeit den H-Index zu berechnen ist es die Publikationen in absteigender Reihenfolge nach der Zitations-Häufigkeit zu sortieren (siehe Tabelle 1).

Publikation	**Zitiert von**
1	50
2	10
3	10
4	7
5	6
6	5
7	1
8	0
9	0

Tabelle 1: Berechnung des H-Index

Nun wird durchgezählt, bis die r-te Publikation weniger als r mal zitiert wurde. Der H-Index ist dann r-1. Im hier beschriebenen Fall (siehe Tabelle 1) ist der H-Index gleich 5, weil 5 Publikation häufiger als 5 mal zitiert wurden und die restlichen weniger als 5 mal.
Der G-Index, welcher in nachfolgendem Unterkapitel erläutert wird, greift die Vorteile des H-Index auf und ergänzt diese um die positive Berücksichtigung herausragender Arbeiten.

2.3. G-Index

Der G-Index wurde von Leo Egghe in seinem Artikel "Theory and practice of the g-index" 2006 eingeführt [4]. Der G-Index bezeichnet gegenüber dem H-Index die Anzahl der Veröffentlichungen mit durchschnittlich g Zitierungen [18]. Er hat den Zweck, den Top-Publikationen, die sehr häufig zitiert wurden, mehr Bedeutung zuzuordnen. Dies ist darauf begründet, dass diese durch häufiges Zitieren mehr literarische Bedeutung haben als die weniger häufig zitierten Publikationen. Der H-Index behandelt hingegen alle Artikel gleich. Es gilt:

$G - Index \geq H - Index$

Auf das Beispiel in Tabelle 1 angewendet bedeutet dies, dass der Wissenschaftler/Forscher einen G-Index von 9 hat. Dieser höhere Wert des G-Index (im Vergleich zum H-Index von 5) kann dadurch erklärt werden, dass häufig zitierte Publikationen auf positive Art und Weise in den G-Index einfließen. Dies ist möglich, da sie die geringere Zitationsrate anderer Publikationen nivellieren.

Um die Daten von Google Scholar als Grundlage für die Berechnungen verwenden zu können ist es notwendig, sie wie im nachfolgenden Kapitel beschrieben, mit Hilfe eines Parsers zu parsen.

2.4. Parser

Ein Parser ist ein Programm, das ein beliebiges Dokument zerlegt und die enthaltenen Informationen in ein für die Weiterverarbeitung brauchbares Format umwandelt. Diese veränderten Informationen werden dann der darüberliegenden Schicht zur Verfügung gestellt. Es wird unterschieden zwischen validierenden und nicht-validierenden Parsern. Nicht-validierende Parser müssen das Dokument nur auf ihre Wohlgeformtheit überprüfen. Validierende Parser müssen zudem die Gültigkeit eines XML-Dokuments überprüfen und jeden Verstoß melden [1].

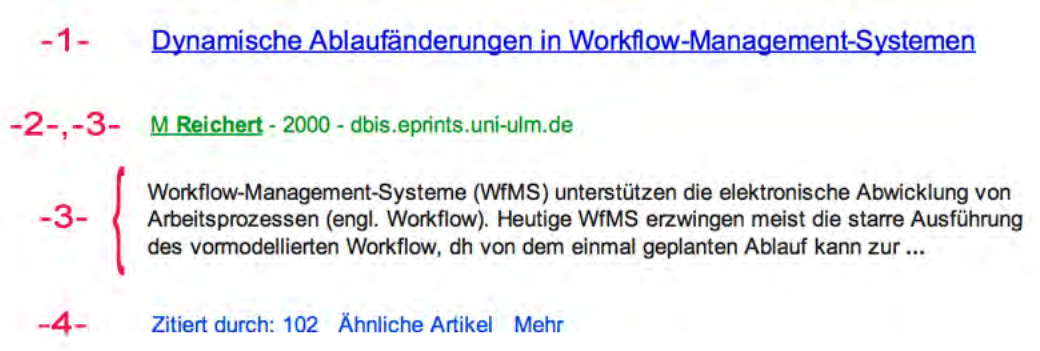

Abbildung 3: Treffer bei Google Scholar

Wenn bei Google Scholar zum Beispiel nach "Manfred Reichert" gesucht wird, enthält der Treffer (siehe Abbildung 3) folgende für uns relevante Daten:

- Titel (- 1 -)
- Autor (- 2 -)
- Datum der Veröffentlichung (- 3 -) mit den Informationen Wo (optional) und Wann (optional)
- Beschreibung (-4 -)
- Anzahl der Zitate (- 5 -)

Diese Daten gilt es für das System in geeigneter Art und Weise zu parsen. Die Daten liegen nicht in grafischer Form vor, sondern in einem Hypertext Markup Language (HTML) Dokument, das wie in Listing 1 aufgebaut ist und folgende Struktur aufweist:

1. Jeder Eintrag befindet sich in einem DIV der Klasse *gs_r*

2. Jeder Eintrag beinhaltet eine DIV Klasse *gs_rt* und ein inneres H3 Element
   a) Das H3 Element beinhaltet A oder SPAN oder beides
      i. Falls beide Elemente enthalten sind, ist die Klasse des SPAN *gs_ctc*
      ii. Falls das Element A nicht enthalten ist, ist die Klasse des SPAN *gs_ctu*

3. Der SPAN mit den Klasse *gs_ggs gs_fl* ist optional
   a) Falls SPAN existiert, beinhaltet er ein A Elemente und ein SPAN Element
   b) Das Element A ist von der Klasse *yC_*
      i. In der *yC_* wird das Blank durch einen Zähler ersetzt.
      ii. Der Zähler zählt alle Elemente des gesamten Dokumentes der Klassen *yC_*

4. Jedes Ergebnis ist in der Klasse *gs_ri*
   a) Das Element A der Klasse *gs_rt* enthält den Titel
   b) Das Element A der Klasse *gs_a* enthält die Webadresse, die Autoren, und die Veröffentlichung
   c) Die Klasse *gs_rsl* enthät die Beschreibung
   d) Das Element A der Klasse *gs_fl* enthält die Anzahl der Zitate

```
1 <div class="gs_r">
2   <div class="gs_ggs gs_fl">
3     <a href="http://dbis.eprints.uni-ulm.de/433/1/PhD_Thesis_Reichert.
          pdf" class=yC1>
4     </a>
5   </div>
6   <div class="gs_ri">
7     <h3 class="gs_rt">
8       <a href="http://dbis.eprints.uni-ulm.de/433/" class=yC0>Dynamische
```

```
9           Ablaufänderungen in Workflow-Management-Systemen
10        </a>
11      </h3>
12      <div class="gs_a">
13        <a href="/citations?user=BHDNcesAAAAJ&amp;hl=de&amp;oi=sra">M
14        <b>Reichert</b> </a> - 2000 - dbis.eprints.uni-ulm.de
15      </div>
16      <div class="gs_rs">
17        Workflow-Management-Systeme (WfMS) unterstützen die elektronische
18        Abwicklung von <br>Arbeitsprozessen (engl. Workflow). Heutige
19        WfMS erzwingen meist die starre Ausführung
20          <br> des vormodellierten Workflow, dh von dem einmal geplanten
              Ablauf kann zur <b> ...</b>
21      </div>
22      <div class="gs_fl">
23        <a href="/scholar?cites=11227125687828591289&amp;as_sdt=2005&amp;
              sciodt=0,5&amp;hl=de">Zitiert durch: 102
24        </a>
25        <a href="/scholar?q=related:ubKvR1TDzpsJ:scholar.google.com/&amp;
              hl=de&amp;as_sdt=0,5">Ähnliche Artikel
26        </a>
27        <a onclick="return gs_ocit(event,'ubKvR1TDzpsJ','0')" href="#"
              class="gs_nph">Zitieren
28        </a>
29        <a href="/scholar?q=info:ubKvR1TDzpsJ:scholar.google.com/&amp;
              output=instlink&amp;hl=de&amp;as_sdt=0,5&amp;scillfp
              =10382554563620473189&amp;oi=llo" class="gs_md_li">
              Bibliothekssuche
30        </a>
31      </div>
32    </div>
33  </div>
```

Listing 1: HTML-Dokument der Google Scholar Trefferliste

Diese Struktur ist schwer zu parsen, da sie dynamisch aufgebaut ist. Das bedeutet, dass nicht immer alle Elemente auftreten müssen. Trotzdem muss ein Algorithmus implementiert werden, der damit umgehen kann.

In dieser Arbeit wird ein Jericho HTML Parser verwendet [10]. Er ist eine Open-Source Java Bibliothek, die es ermöglicht Teile eines HTML Dokuments zu analysieren und zu manipulieren.

2.5. Web Service

Ein Web Service ist eine im World Wide Web (WWW) zur Verfügung gestellte Komponente, die eine Abstraktionsebene einer Anwendungslogik darstellt. Auf den Dienst kann über einfache Internetstandardprotokolle zugegriffen werden. Eine hohe Verfügbarkeit und einfache Bereitstellung von Diensten sind Ziele eines Web Services. Sie können auf verschiedene Arten implementiert werden. Da es verschiedene Anforderungen in der Web-Kommunikation gibt, haben sich die beiden Programmierparadigmas SOAP und REST herausgebildet. Diese beiden haben eine herausragende Stellung, da sie den Begriff *Web Service* am meisten geprägt haben [17]. SOAP hat gegenüber REST einen zeitlichen Vorsprung, was dazu geführt hat, dass ein Web Service häufig mit SOAP assoziiert wird.

Simple Object Access Protocol (SOAP)

Die erste Version dieses Netzwerkprotokolls wurde im Jahr 1998 entwickelt. Bis heute wurde es immer weiterentwickelt und verbessert. Zudem ist SOAP seid 2003 ein industrieller Standard des World Wide Web Consortiums (W3C) [15].

Bei der Nachrichtenübermittlung stützt sich SOAP auf die Dienste anderer Standards. Zur Repräsentation der Daten nutzt er Extensible Markup Language (XML) und zur Übertragung der Nachrichten Internet-Protokolle. Die gängigste Kombination SOAP zu realisieren ist die Verwendung von Hypertext Transfer Protocol (HTTP) und Transmission Control Protocol (TCP). Zudem können SOAP-Anfragen zum Beispiel auch über Java Messaging Service (JMS) oder über die File Transfer Protocol (SMTP)/POP3-E-Mail-Protokolle versandt werden.
Bei der Kommunikation zwischen zwei Partnern ruft SOAP entfernte Operationen auf, die die Parameter und Ergebnisse in XML kodieren und übertragen [17]. SOAP ist unabhängig von Betriebssystemen, Programmiersprachen und Objektmodellen und kann verschiedene Plattformen miteinander verbinden [12].
Die wichtigsten Vorteile von SOAP sind eine allgemein akzeptierte Standardisierung, Plattformunabhängigkeit, Offenheit, Robustheit und Skalierbarkeit. Der gravierendste Nachteil hingegen ist viel Overhead und dadurch eine etwas geringere Performance wegen des verwendeten Nachrichtenformats XML. Daher wird im nachfolgenden Unterkapitel eine Alternative zu SOAP vorgestellt.

Representational State Transfer (RESTful)

Entwickelt wurde die REST-Architektur von Fielding im Jahr 2000 im Rahmen seiner Dissertation zu den Erfolgen des WWW [5]. Nach Ingo Melzer (2007): "REST ist kein ei-

genes Protokoll, sondern ein Architekturstil, der definiert, wie existierende Web-Protokolle verwendet werden können, um unter anderem möglichst einfache Web Services zu realisieren.” [12]. Beim RESTful Web Service wird die Anfrage meist über eine HTTP-Anfrage an den Web-Server gestellt.
Die Ressource ist über eine Uniform Resource Locator (URL) kodiert [17]. Bei REST gibt es nur wenige Operationen, die unterstützt werden, wovon zwei das Lesen und Aktualisieren sind. Im Mittelpunkt von REST steht die Ressource, die eindeutig adressierbar ist. Da die Vorstellungen von REST auseinander gehen, gibt es fünf Eigenschaften, die nach Fieldinger erfüllt werden müssen [5] .

1. *Adressierbarkeit:*
   Jede Ressource muss über einen eindeutigen Unique Resource Identifier (URI) identifiziert werden können.

2. *Zustandslosigkeit:*
   Die Kommunikation der Teilnehmer untereinander ist zustandslos. Dies bedeutet, dass keine Benutzersitzungen (etwa in Form von Sessions und Cookies) existieren, sondern bei jeder Anfrage alle notwendigen Informationen wieder neu mitgeschickt werden müssen. Durch die Zustandslosigkeit sind RESTful Web Services sehr einfach skalierbar.

3. *Einheitliche Schnittstelle:*
   Auf jede Ressource muss über einen einheitlichen Satz von Standardmethoden zugegriffen werden können, wie zum Beispiel Lesen, Schreiben, Erzeugen und Löschen. Ähnlich wie die Standard-HTTP-Methoden wie GET, POST, PUT und weitere.

4. *Verschiedene Repräsentationen:*
   Dies bedeutet, dass verschiedene Repräsentationen einer Ressource existieren können. Ein Client kann somit etwa eine Ressource explizit, beispielsweise im HTML, XML, JSON-Format anfordern, oder sogar die Sprache wählen.

5. *Verwendung von Hypermedia*
   Einfache Navigation von einer Ressource zu einer anderen ohne umständliche Registrierungs-Datenbanken oder ähnliche Infrastrukturen.

Ist eine Anwendung konform zum REST-Architekturstil, wird sie als RESTful bezeichnet [12]. Um die Daten plattformunabhängig zur Verfügung zu stellen, wird ein einfach zu serialisierendes und zu de-serialisierendes Datenformat verwendet, welches im folgenden Kapitel erläutert wird.

2.6. JSON

JSON ist ein kompaktes Datenformat, das dazu dient Daten auszutauschen. Es ist sehr einfach aus Klammern (geschweift/eckig), Doppelpunkten und Kommas aufgebaut. Aus diesem Grund ist es einfach zu lesen, zu interpretieren und zu parsen. Es baut auf dem Standard JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999 [2] auf. Da JSON textbasiert ist, ermöglicht es einen strukturierten Datenaustausch zwischen allen Programmiersprachen [3]. Eine JSON Datei ist aus den Werten Objekte, Array, Zahlen, Zeichenketten, Boolische Werte und dem Nullwert aufgebaut.(Siehe Abbildung 4) Diese werden nachfolgend erläutert.

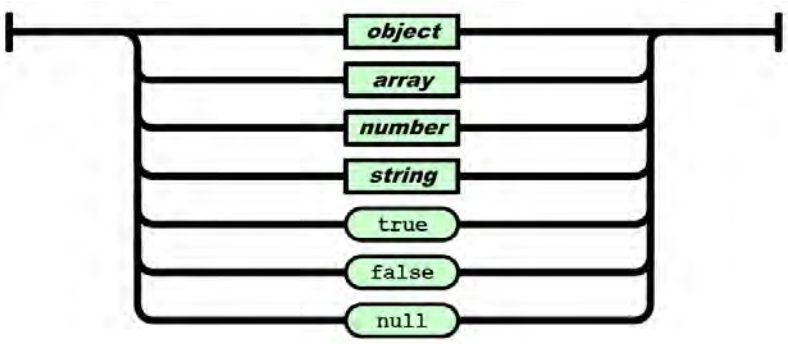

Abbildung 4: JSON Value [3]

Objekte

Ein Objekt ist ein Paar geschweifter Klammern, die 0 bis n weitere Wertepaare (aus Name und value) einschließen, wobei der Name ein String ist, der immer eindeutig sein muss. Ein Doppelpunkt trennt das (Name:value)-Paar voneinander und ein darauffolgendes Komma trennt wiederum die Werte voneinander.

Array

Ein Array ist ein Paar aus eckigen Klammern, die 0 bis n weitere Werte einschließen. Die Werte des Arrays werden durch ein Komma getrennt.

Zahlen

Der Wert Zahlen bezeichnet die Folge der Ziffern 0–9. Diese Folge kann durch ein negatives Vorzeichen (-) eingeleitet und durch einen Dezimalpunkt (.) unterbrochen werden. Die Zahl kann durch die Angabe eines Exponenten (e) oder (E) (siehe Abbildung 5) ergänzt werden, dem ein Vorzeichen (+) oder (-) und eine Folge der Ziffern 0–9 folgt.

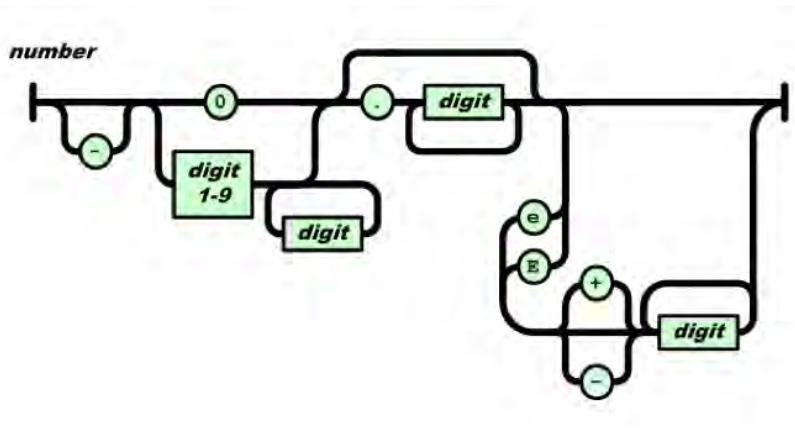

Abbildung 5: JSON Number [3]

Zeichenketten

Eine Zeichenkette, wie in Abbildung 6 veranschaulicht, beginnt und endet mit doppelten geraden Anführungszeichen (”). Sie kann Unicode-Zeichen und Escape-Sequenzen enthalten.

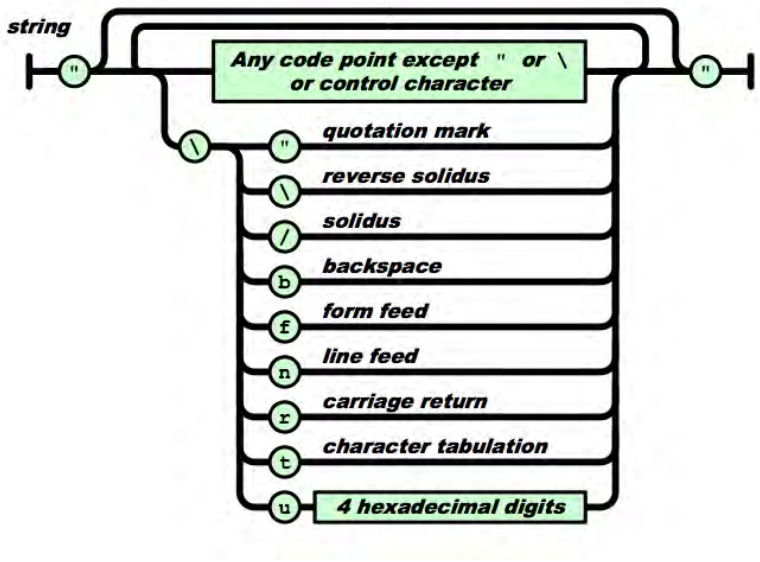

Abbildung 6: JSON String [3]

Boolische Werte

Sie werden durch die zwei Schlüsselwörter true oder false dargestellt.

Nullwert

Der Nullwert wird durch eine Null dargestellt.

3. Anforderungsanalyse

Nach Sommerville 2012 beschreiben Anforderungen in der Systementwicklung Dienste und Bedingungen, die ein System oder eine Systemkomponente leiten und erfüllen soll, um ein Problem zu lösen oder zu erreichen [16]. Hierbei wird zwischen funktionalen und nichtfunktionalen Anforderungen unterschieden.
Funktionale Anforderungen beschreiben dabei Dienste oder die Funktionalitäten, die ein System bereitstellen soll. **Nichtfunktionale Anforderungen** dagegen beschreiben Beschränkungen der Dienste oder Funktionalitäten des Systems.
Diese Anforderungen werden in den nächsten Kapiteln sowohl für den Web Service als auch für die App definiert.

3.1. Web Service

In diesem Unterkapitel werden die funktionalen Anforderungen und anschließend die nichtfunktionalen Anforderungen an den Web Service definiert und beschrieben.

Funktionale Anforderungen

- **FA#1: Einfache Suche bei Google Scholar**
  Beschreibung: Suche in Google Scholar nach einem Parameter.
- **FA#2: Erweiterte Suche bei Google Scholar**
  Beschreibung: Suche in Google Scholar nach mehreren Parametern.
- **FA#3: Parsen der Daten**
  Beschreibung: Daten von Google Scholar müssen geparst werden, damit sie in einem geeigneten Format zur Verfügung stehen.
- **FA#4: Berechnung des H-Index**
  Beschreibung: Berechnung des H-Index der jeweiligen Anfrage.
- **FA#5: Berechnung des G-Index**
  Beschreibung: Berechnung des G-Index der jeweiligen Anfrage.
- **FA#6: Speicherung der Daten in einer Datenbank**
  Beschreibung: Bei der einfachen Suchanfrage an Google Scholar sollen die Daten der Suche für 30 Tage gespeichert werden. Bei einer erneuten Anfrage werden die Daten zurückgegeben.

- **FA#7: Berechnung der Zeitdifferenz des letzten Aufrufs**
  Beschreibung: Bei jedem Aufruf soll ein Zeitstempel eingespeichert werden, mit dem berechnet werden kann, wie alt der Eintrag in der Datenbank ist.

- **FA#8: Löschen nicht mehr benötigter Daten**
  Beschreibung: Gespeicherte Daten in der Datenbank, die älter als 30 Tage sind, sollen automatisch gelöscht werden.

- **FA#9: Umgehung der Limitation der Zugriffe auf Google Scholar**
  Beschreibung: Da Google Scholar die Zugriffe auf Scholar limitiert, sollen diese durch Einsetzen von Proxyservern umgangen werden.

- **FA#10: Intelligente Wahl des Proxyservers**
  Bechreibung: Proxyserver sollen zufällig und nach der Quantität (wie schnell sie auf eine Anfrage reagieren) ausgesucht werden.

- **FA#11: Bewertung der Proxyserver**
  Beschreibung: Proxyserver sollen nach jeder Benutzung nach ihrer Antwortzeit bewertet werden.

- **FA#12: Löschen von Proxyservern**
  Beschreibung: Mehrmals zu langsame Proxyserver sollen gelöscht werden.

Nichtfunktionale Anforderungen

- **NFA#1: Schnelle Reaktion auf Anfragen**
  Beschreibung: Der Web Service soll schnellstmöglich auf Anfragen reagieren.

- **NFA#2: Wartbarkeit der Datenbank**
  Beschreibung: Die verwendete Datenbank sollte gut zu warten sein.

- **NFA#3: Wie lange Daten gecacht werden**
  Beschreibung: Die Dauer des Cachens sollte leicht einzustellen sein.

3.2. DBIS Scholar App

Die funktionalen und nichtfunktionalen Anforderungen an die App werden in diesem Unterkapitel definiert und beschrieben.

Funktionale Anforderungen

- **FA#1: Einfache Suche**
  Beschreibung: Die App sollte eine einfache Suchanfrage ermöglichen.
- **FA#2: Erweiterte Suche**
  Beschreibung: Die App sollte eine erweiterte Suche mit allen Parametern ermöglichen.
- **FA#3: Speicherung der Parameter vergangener Suchen**
  Beschreibung: Es sollen die Daten der letzten 10 Suchanfragen gespeichert werden. Zudem sollen diese für eine erneute Suche zur Verfügung stehen.
- **FA#4: Anzeigen der Datenliste**
  Beschreibung: Die Daten sollen in einer Tabelle angezeigt werden.
- **FA#5: Detaillierte Anzeige eines Treffers**
  Beschreibung: Auswahl und Anzeigen eines Eintrages in der Datenliste.
- **FA#6: Drucken der Daten**
  Beschreibung: Es soll möglich sein, alle Daten zu drucken.
- **FA#7: Versenden der Daten**
  Beschreibung: Es soll möglich sein, die Daten per PDF zu verschicken.
- **FA#8: Kopieren der Daten/Weblinks**
  Beschreibung: Datenlisten und Weblinks sollen kopierbar sein.
- **FA#9: Anzeigen der Website des Treffers**
  Beschreibung: Es soll möglich sein, sich die Website des Treffers anzeigen zu lassen.

- **FA#10: Anzeigen der berechneten bibliometrischen Maße**
  Beschreibung: Der H-Index, G-Index und die Anzahl der gesamten Zitate sollen angezeigt werden.

- **FA#11: Suche innerhalb der Datenliste**
  Beschreibung: Die Datenliste soll durch eine Suchfunktion durchsuchbar sein.

Nichtfunktionale Anforderungen

- **NFA#1: Navigation in der App**
  Beschreibung: Die Navigation in der App soll einfach und verständlich sein.

- **NFA#2: Anzeigen von Aktivitäten der App**
  Beschreibung: Alle Aktivitäten der App sollen angezeigt werden.

- **NFA#3: Die App soll ab iOS 6 universell laufen**
  Beschreibung: Die App soll ab iOS6 für die Geräte iPad, iPhone und iPod laufen.

4. Implementierung

Das gesamte System beseht aus den vier Hauptkomponenten **Anwender, Web Service, Proxyserver und Google Scholar** (siehe Abbildung 7). Der Anwender ist dabei ein Endgerät/Benutzer. Der Web Service ist ein System, das verschiedene Funktionen für die Suche zur Verfügung stellt und die angefragten Dateien anonym speichert. Der Proxyserver ist für die Verschleierung der Internetprotokoll (IP)-Adresse zuständig und Google Scholar ist die Suchmaschine für Literatur.
Durch dieses Architektur-Konzept ist das Erstellen der App auf anderen Software-Plattformen erleichert, da zentrale Funktionen, wie parsen, cachen und die Umgehung der Limitation an Zugriffe auf Google Scholar durch Proxyserver, vom Web Service für alle Anwender angeboten werden.

Der Ablauf einer Anfrage an den Web Service sieht folgendermaßen aus:

1. Der Anwender stellt eine HTTP Anfrage an den Web Service, der über eine eindeutige URI erreichbar ist.
2. Der Web Service leitet die Anfrage dann über einen Proxyserver weiter an Google Scholar.
3. Google Scholar generiert die Antwort und leitet die Trefferliste an den Web Service als HTML Dokument zurück.
4. Der Web Service parst die Informationen aus dem Dokument und sendet sie an den Anwender per JSON zurück.

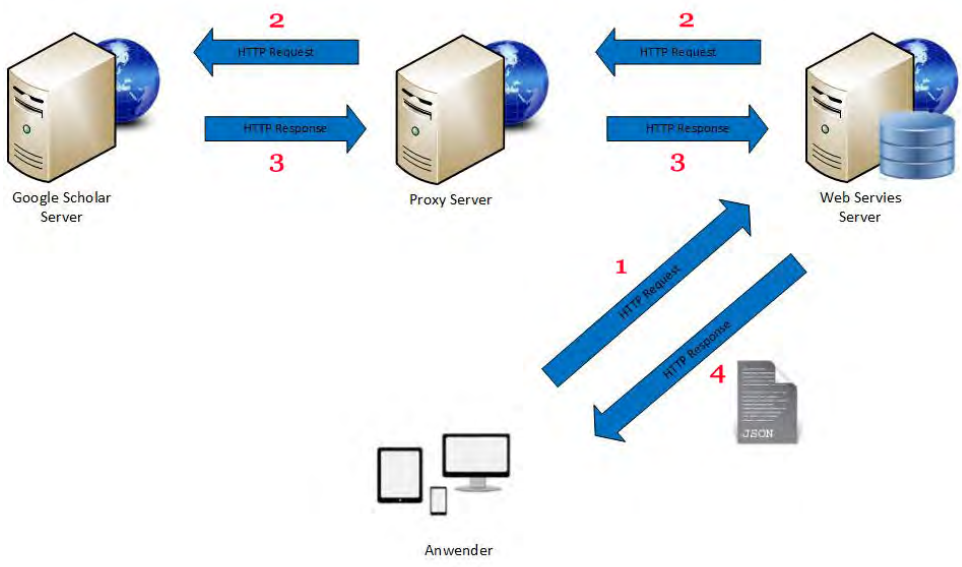

Abbildung 7: Ablauf einer Anfrage

4.1. Google Scholar

Die beiden Suchmethoden *einfache Suche* und *erweiterte Suche* stehen bei Google Scholar bei den zwei folgenden URL's zur Verfügung:

1. http://scholar.google.de/scholar?hl=de&q=&btnG=&lr=
2. http://scholar.google.de/scholar?as_q=&as_epq=&as_oq=&as_eq=&as_occt=any&as_sauthors=&as_publication=&as_ylo=&as_yhi=8&btnG=&hl=de&as_sdt=0%2C5

Die URL's können dabei, je nachdem nach welchen Einschränkungen (Suchparametern) gesucht werden soll, beliebig zusammengestellt werden (siehe Tabelle 2).

Suchparameter	**Beschreibung**	**Typ**
as_q	mit allen Wörtern	Text
as_epg	mit der genauen Wortgruppe	Text
as_oq	mit irgendeinem der Wörter	Text
as_eq	ohne die Wörter	Text
as_occt	die meine Wörter enthalten	any oder many
as_sauthors	die von folgenden Autoren veröffentlicht wurden	Text
as_publication	Artikel zurückgeben, die hier veröffentlicht wurden	Zahl
as_ylo und as_yhi	Artikel zurückgeben, die in folgendem Zeitraum geschrieben wurden	Zahl

Tabelle 2: Such-Parameter

Zudem ist es möglich, durch den URL weitere Einstellungen vorzunehmen, wie die Sprache und die Anzahl der angezeigten Treffer pro Seite (siehe Tabelle 3).

Suchparameter	**Beschreibung**	**Typ**
num	Anzahl der Treffer pro Seite	10, 20
hl	Sprache	de (Deutsch), en (Englisch)

Tabelle 3: Mögliche zusätzliche Einstellungen über den URL

4.2. Proxyserver

Google Scholar analysiert die Anfragen einer IP-Adresse. Bei zu häufigen und denselben Anfragen von einer IP-Adresse, wird der Account bzw. die IP-Adresse gesperrt. Dies muss mithilfe von Proxyservern umgangen werden.
Das System wählt dafür zufällig einen Proxyserver aus einer Datenbank aus und stellt über diesen die Anfrage an Google Scholar. Bei jeder Anfrage wird die Zeit, die der Proxyserver zum Antworten benötigt, gemessen. Anhand dieser Zeit wird die Qualität als Integer (1 = gut, 2 = zweite Chance, 3=schlecht) des Proxyservers gespeichert.

```
1  if (((zstNachher - zstVorher) / 1000) > 15) {
2
3    DatabaseUpdate update = new DatabaseUpdate();
4
5      if (proxyentry.getQuality() == 1) {
6        proxyentry.setQuality(2);
7      } else {
8        proxyentry.setQuality(3);
9      }
10   update.updateProxyList(proxyentry);
11 }
```

Listing 2: Bewertung der Proxyserver

1. Falls der Proxyserver als *gut* bewertet wird, wird er weiterhin verwendet.
2. Falls er als *zweite Chance* bewertet wird, wird er zu einem späteren Zeitpunkt nochmals verwendet und nochmals getestet. Falls er wieder zu langsam ist, wird er mit *schlecht* bewertet.
3. Falls er als *schlech* bewertet wird, wird er nicht mehr verwendet und wird beim cleanup gelöscht.

Dadurch wird gewährleistet, dass immer ein schneller (qualitativ hochwertiger) Proxyserver zur Verfügung steht.

4.3. Web Service

Der Web Service besteht aus drei Komponenten. Diese sind der **Web Service**, die **Logik Einheit** und die **Datenbank** (siehe Abbildung 8). Der Web Service fungiert als Vermittler zwischen dem Anwender und Google Scholar. Die Logik Einheit ist dafür zuständig, die Daten die von Google Scholar zurückkommen, aufzubereiten, in der Datenbank zu speichern und dem Web Service wieder zur Verfügung zu stellen.

Abbildung 8: Übersicht über den Web Service

4.3.1. Logik Einheit

Die Logik Einheit ist zuständig für die Steuerung der angebotenen Funktionen des Web Service. Sie steuert die Anfrage an Google Scholar und übernimmt das Parsen der Daten und viele weitere Funktionen. Um diese Aufgaben zu realisieren, besteht die Logik Einheit aus den neun Klassen: *Bookentry, Booklist, Bookindex, Proxyentry, Proxylist, Parameter, Request und ExtractText*. Das Klassendiagramm (siehe Abbildung 9) zeigt die Zusammenhänge der einzelnen Klassen und deren jeweilige Attribute und Methoden. Um sich den Ablauf einer Anfrage klar zu machen, ist es erforderlich, das Klassendiagramm der Logik Einheit zu verstehen.

Klassen

Die Klasse *Bookentry* beinhaltet die relevanten Daten (siehe Kapitel 2.4) des Treffers. Sie bietet die Methode *CompareTo()*, mit der die Anzahl der Zitate verglichen werden kann.

Die Klasse *Booklist* kann sowohl den G-Index als auch den H-Index berechnen. Zudem kann sie Bücherlisten und Bucheinträge hinzufügen. Mithilfe der Methode *CompareTo()* kann sie die *Booklist* nach Häufigkeit der Zitate sortieren.

Die Klasse *Bookindex* speichert die relevanten Daten über den Index des Treffers und bietet die Methode *calculateTimeDiff()*. Sie ermöglicht es, die Zeitdifferenz zwischen dem Datum, an dem die Daten gespeichert wurden und dem heutigen Tag zu berechnen. Damit wird die **FA#3: Berechnung der Zeitdifferenz des letzen Aufrufs** des Web Service realisiert.

Die Klasse *Proxyentry* beinhaltet alle Daten, die für eine Verbindung zu einem Proxyserver benötigt werden.

Die Klasse *Proxylist* bietet eine Methode an, um aus der Proxylist einen Proxyserver anhand seiner Qualität (siehe Kapitel 4.2) auszuwählen. Dadurch wird die **FA#10: Intelligente Wahl des Proxyservers** des Web Services verwirklicht.

Die Klasse *Parameter* stellt nur zwei Parameter zur Verfügung, mit welchen entweder eine einfache oder erweiterte Suche aufgerufen werden kann.

Die Klasse *Hitlist* generiert eine Liste, die aus einer Booklist und einem BookIndex besteht. Sie bietet die Methode *saveHitlist()* zum Speichern der Hitlist in der Datenbank an.

Die Klasse *Request* ist zuständig für die Anfragen an Google Scholar. Sie kann eine einfache Anfrage (**FA#1: Einfache Suche bei Google Scholar** des Web Service) und eine erweiterte Anfrage (**FA#2: Erweiterte Suche bei Google Scholar** des Web Service) an Google Scholar stellen.

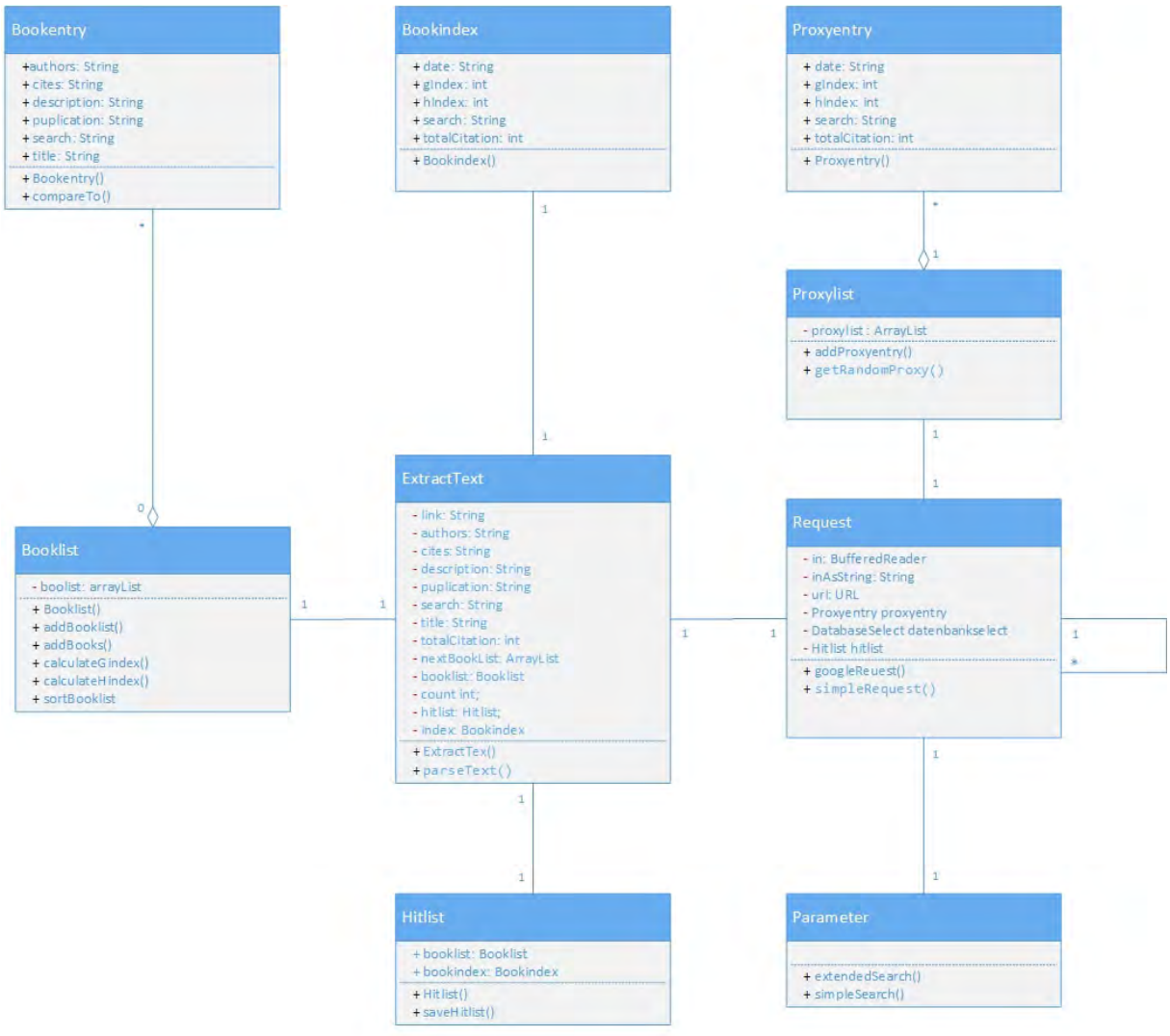

Abbildung 9: Klassendiagramm der Logik Komponente

Die Klasse *ExtractText* ist zuständig für das Parsen der Daten (**FA#3: Parsen der Daten** des Web Service) und das erneute Anfragen an Google Scholar, falls weitere Daten vorliegen. Dies ist nötig, da Google Scholar in der Grundeinstellung nur 10 Treffer pro Seite anzeigt. Das lässt sich aber durch eine Veränderung der URL auf 20 erhöhen. Trotzdem ist es nötig, dass der Web Service Seite für Seite aufruft und die benötigten Daten aus der HTML Datei parst. Zudem ist sie zuständig für das Speichern der Daten und die Berechnung der Indices, bzw. für den Methodenaufruf der beteiligten Klassen.

Abbildung 10 zeigt den Ablauf einer Anfrage der Logik Einheit. Es wird entweder ein einfacher Parameter (Einfache Suche) aufgerufen, oder ein erweiterter Parameter (Erweiterte Suche). Je nachdem wird ein *Simple request* oder ein *Google request* aufgerufen.

1. Falls ein *Simple request* aufgerufen wird, wird überprüft, ob die Daten in der Datenbank vorhanden sind.
   a) Falls ja, wird überprüft, ob die Daten älter als 30 Tage sind.
      i. Sind die Daten älter als 30 Tage, werden die Daten gelöscht und es wird ein *Google request* aufgerufen.
      ii. Falls das nicht der Fall ist, werden die Daten geladen, dem Web Service übergeben und die Anfrage beendet.
   b) Falls nicht, wird ein *Google request* aufgerufen.

2. Falls ein *Google request* aufgerufen wird, werden die Daten bei Google Scholar angefragt und in *ExtractText* geparst. Anschließend wird untersucht, ob noch weitere Seiten zum Parsen vorliegen:
   a) Falls nicht wird überprüft, ob die Anfrage von einem Simple Request kam oder nicht.
      i. Falls ja, werden die Daten in einer Datenbank gespeichert.
      ii. Falls nicht werden die Daten übergeben und das Programm beendet die Anfrage.
   b) Falls ja, wird ein neuer *Google request* mit der folgenden Seite aufgerufen und erneut in *ExtractText* geparst. Anschließend wird überprüft, ob es noch weitere Seiten gibt.

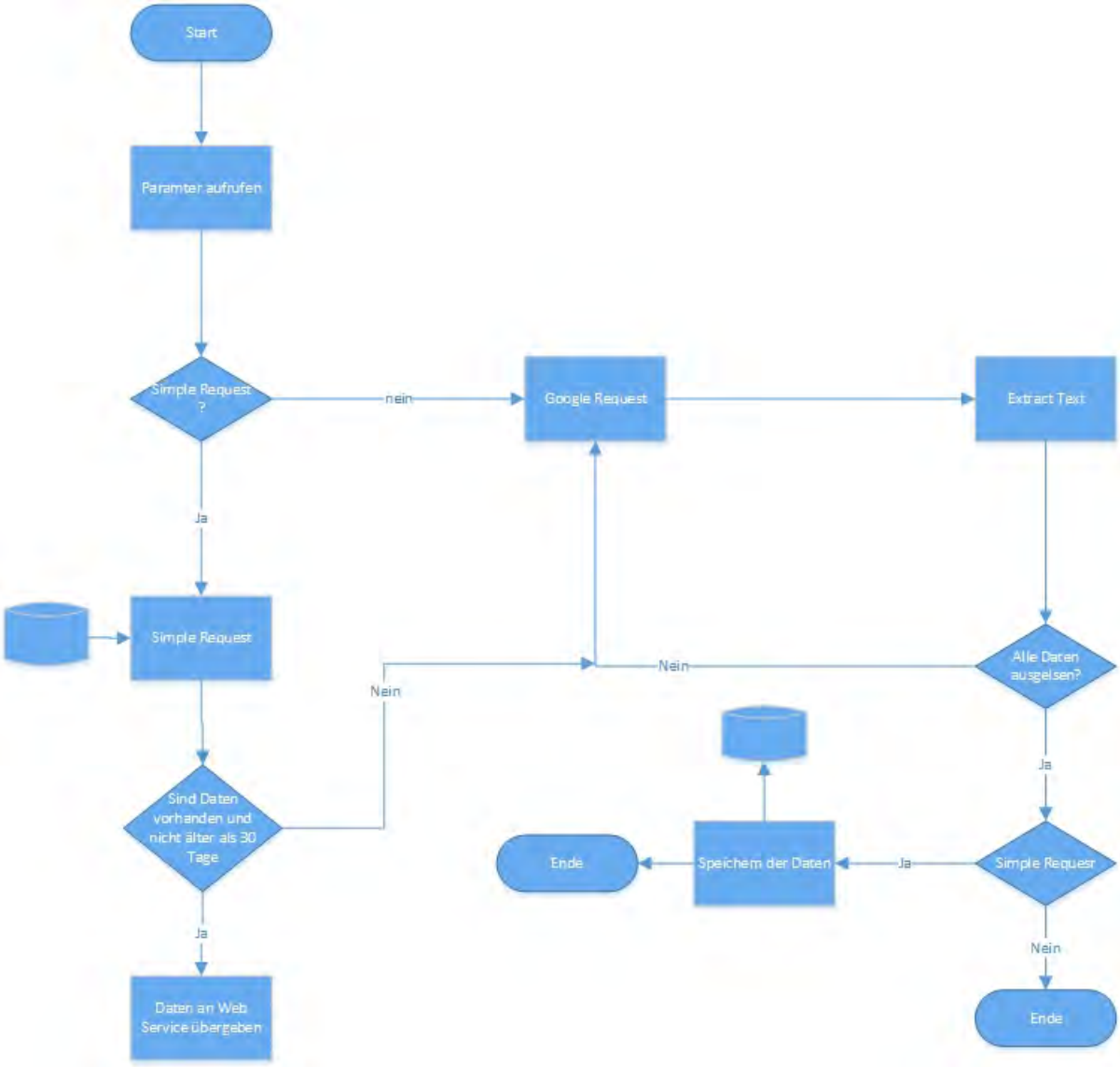

Abbildung 10: Detaillierter Ablauf einer Anfrage

Nun wird die Methode *parseText()* der Klasse *ExtractText* betrachtet, die für das Parsen des Textes und für das Speichern der Daten zuständig ist. Dadurch werden die **FA#3: Parsen der Daten** und **FA#3: Speichern der Daten** des Web Services realisiert.

Wird zum Beispiel in Google Scholar nach *Manfred Reichert* gesucht, sieht der entsprechende URL folgendermaßen aus:

$http://scholar.google.de/scholar?hl=de\&q=manfred+reichert\&btnG=\&lr=$

Der Suchbegriff der Suche wird dann im oberen Teil der Seite dargestellt (siehe Punkt 1 in Abbildung 11).

Im mittleren Teil der Seite befindet sich die Trefferliste (siehe Punkt 2 in Abbildung 11). Diese wurde aus Platzgründen auf drei Treffer reduziert.

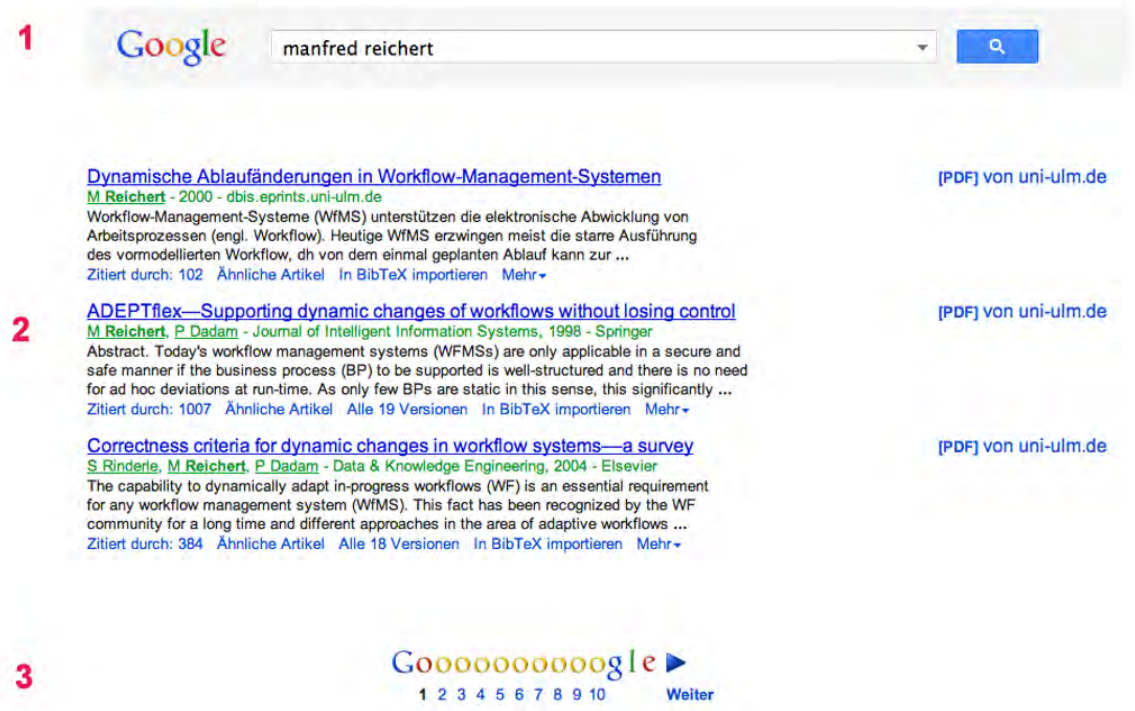

Abbildung 11: Aufbau von Google Scholar [8]

Google Scholar zeigt standardmäßig pro Ergebnisseite 10 Treffer an. Die Anzahl der angezeigten Treffer pro Seite kann durch eine Veränderung in der URL auf 20 erhöht werden. Dies ist, wie bereits erwähnt, über den Befehl „num=20“ in der URL möglich. Es können allerdings maximal 20 Treffer angezeigt werden. Wäre es möglich, sich alle Treffer auf einmal anzeigen zu lassen und folglich alle Treffer auf einmal zu parsen, müsste man sich nicht mit der Navigation der Trefferseiten beschäftigen.

Ganz unten auf der Seite können mit Hilfe einer Navigation (Punkt 3 in Abbildung 11) die verschiedenen Seiten ausgewählt werden, deren Treffer angezeigt werden sollen. Je nachdem wie viele Treffer gefunden wurden, variiert die Anzahl der auszuwählenden Seiten. Die URL jeder Seite beinhaltet die Information, bei welchem Treffer begonnen wird.

Wird zum Beispiel Seite 2, mit der Bedingung, dass 20 Treffer pro Seite angezeigt werden

sollen, aufgerufen, so wird der URL um eine Start (start=20) Information erweitert.

$http://scholar.google.de/scholar?start=20\&q=manfred+reichert\&hl=de\&as_sdt=0,5$

DIe Methode*parseText()* parst die Daten, die in Kapitel 2.1 beschrieben wurden mit den Funktionen die von *Jericho* angeboten werden [10]. Um dies zu veranschaulichen wird in Listing 3 aufgezeigt, wie der *Titel* aus dem HTML-Dokument geparst wird.

Zuerst wird mit der Methode *getAllElements("div class="gs_ri)* eine Liste erstellt, die alle Treffer beinhaltet. Diese Liste wird mit einer *For-Schleife* durchgegangen. Das erste Elemente des h3 class="gs_rt" wird durch die Methode get.getFirstElement("h3 class="gs_rt) ausgewählt und anschließend durch die Methode *getTextExtractor().toString()* der Inhalt extrahiert.

```
1  List<Element> linkElements_erster_div = source.getAllElements("div class
       =\"gs_ri\"");
2
3    for (Element linkElement_erster_div : linkElements_erster_div) {
4      try {
5          titel = linkElement_erster_div.getFirstElement("h3 class=\"gs_rt
               \"").getTextExtractor().toString();
6
7      } catch (Exception e) {
8        // kein Titel gefunden
9        titel = "Notset";
10     }
11   }
```

Listing 3: Parsen des Titels

Alle weiteren Informationen werden nach dem gleichen Muster und nach der im Kapitel 2 vorgestellten Struktur geparst.

Zudem ist es notwendig, dass die Methode in der Lage ist, die einzelnen Trefferseiten selbständig nacheinander aufzurufen. Dazu wird eine ArrayList und eine Zählervariable mitverwaltet. Falls noch weitere Trefferseiten existieren, stellen diese beiden Komponenten einen neuen Request an Google Scholar. In Abbildung 12 ist vereinfacht dargestellt, was die Methode *parsetext()* alles leisten muss. Zu Beginn werden Daten bei Google Scholar angefragt. Sind diese nicht vorhanden, werden, falls zuvor schon Daten hierzu angefragt wurden, diese gespeichert. Sind Daten vorhanden, werden diese geparst. Falls noch weitere Trefferseiten vorliegen und die Zählervariable kleiner als 50 ist, wird die nächste Seite angefragt.

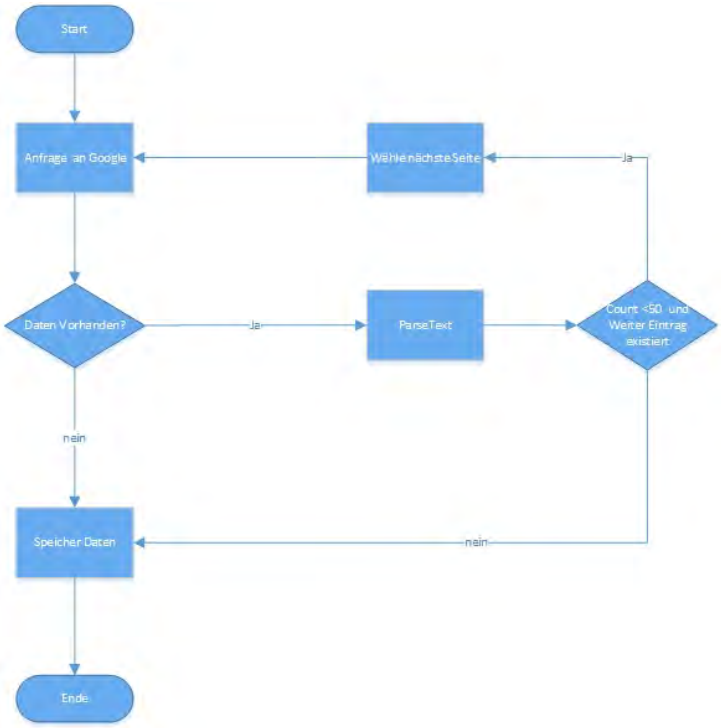

Abbildung 12: Ablauf der Methode *parsetext()*

Im nachfolgenden Kapitel werden die Datenbank und die Methoden, die der Web Service anbietet, um die Daten zu speichern, erläutert.

4.3.2. Datenbank

Um die Daten einer Suchanfrage zu speichern, wird eine MySQL-Datenbank verwendet. Dadurch wird viel Datenverkehr mit Google Scholar vermieden, da die Daten gecacht werden können. Der einzige Nachteil besteht darin, dass die Daten bei zu langer Speicherungszeit nicht mehr aktuell sein könnten.
Die Datenbank besteht aus den Tabellen *Bookindex*, *Booklist* und *Proxylist* (siehe Abbildung 13).

In der Tabelle *Bookindex* werden wichtige Informationen über die Suche gespeichert.

- **search:** Speichert die Suchanfrage unter dem Suchbegriff.
- **date:** Speichert das Datum zum Zeitpunkt des Aufrufes.
- **H-Index:** Speichert den berechneten H-Index der Suche.
- **G-Index:** Speichert den berechneten G-Index der Suche.
- **totalCitation:** Speichert die aufsummierte Anzahl der Zitate der Suche.

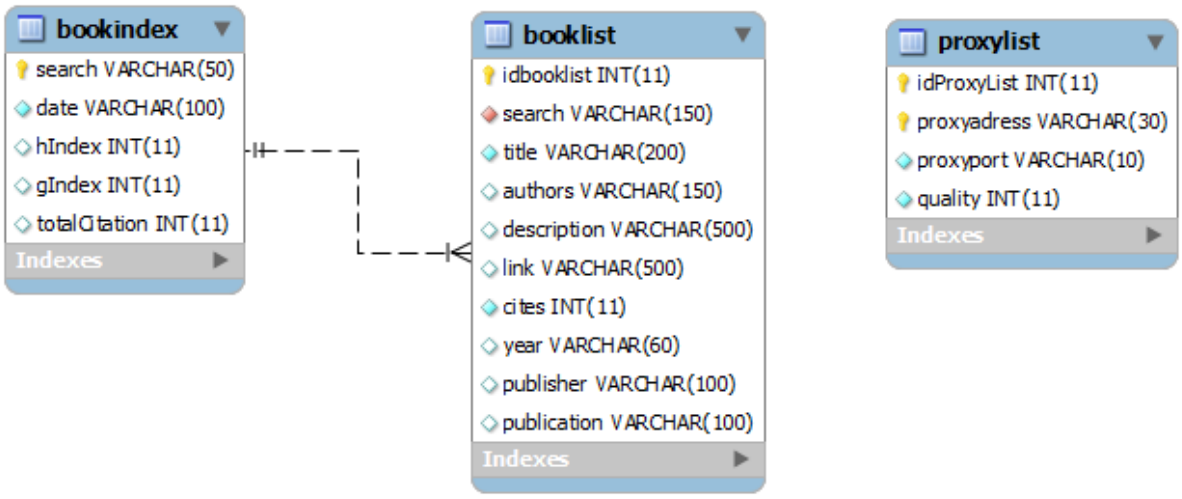

Abbildung 13: Datenbankmodell

Des Weiteren werden in der Tabelle Booklist die einzelnen Informationen über den Treffer gespeichert. Diese werden über den Primärschlüssel *search* identifiziert. Damit der Vorgang nicht so lange dauert, sind die Daten in einer B-Baum-Struktur aufgebaut (siehe Kapitel 4.2.2 B-Baum).

- **search:** Speichert die Suchanfrage unter dem Suchbegriff.
- **title:** Speichert den Titel des Treffers.
- **author:** Speichert den Autor.
- **description:** Speichert die Beschreibung des Treffers.
- **link:** Speichert die Webadresse des Treffers.

Zudem ist es notwendig, eine Liste mit Proxyservern zu verwalten, da Google Scholar nach zu vielen Anfragen den Account bzw. die IP-Adresse für gewisse Zeit sperrt. Diese Sperre wird zwar nach einiger Zeit wieder aufgehoben, aber es gilt mit dieser Tatsache umgehen zu können.

- **proxyadresse:** Speichert die IP-Adresse des Proxyservers.
- **proxyport:** Speichert den Port des Proxyservers.
- **quality:** Speichert die Qualität des Proxyservers, wobei hier die Geschwindigkeit des Proxyservers gemeint ist.

In Abbildung 14 wird ein Auszug des Klassendiagramm der Datenbank mit den wichtigesten Attributen und Methoden dargestellt. Die *DatabaseConnection* ist zuständig für die Verbindung zur Datenbank. Sie beinhaltet alle dazu erforderlichen Informationen, wie zu Beispiel den Datenbanknamen, den Port und den Host. Die Informationen werden an alle anderen Klassen vererbt. Die Klassen *DatabaseSelect, DatabaseInsert, DatabaseUpdate und DatabaseDelete* sind für das Auswählen, Einfügen, Aktualisieren und Löschen von Datensätzen zuständig.

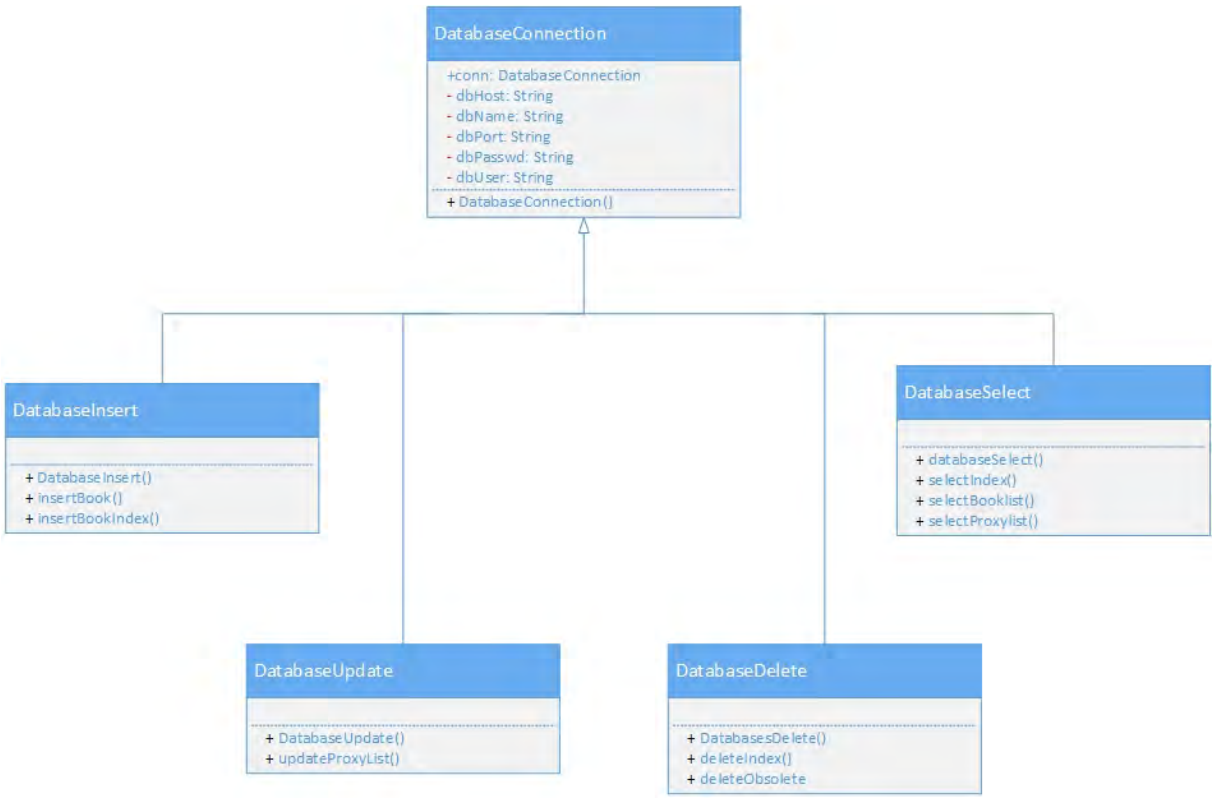

Abbildung 14: Auszug des Klassendiagramm der Datenbank

4.3.3. B-Baum

Wie bereits in Kapitel 4.3.2 erläutert, werden die Daten über die Suche in einem Index und die dazu gehörigen Publikationen in einer extra Tabelle gespeichert. Diese werden über einen Primärschlüssel identifiziert. Da aber zu jedem Index mehrere hundert Publikationen existieren, wird in dieser Arbeit eine B-Baum-Struktur verwendet, die eine schnelle Zuordnung ermöglicht. Dazu werden die Daten nach Schlüsseln sortiert und gespeichert. In dieser Arbeit heißt das, dass durch den B-Baum bekannt ist, welche Publikationen zu welchem Index gehören und umkehrt.

4.4. Anwender

Der Anwender kann unter anderem ein Internet-Browser, ein Android-Gerät oder ein iPhone sein. Da der Web Service plattformunabhängig programmiert ist und er die Daten per JSON liefert, muss sich jede Anwendung selbst darum kümmern, wie sie diese Daten verarbeitet. Das JSON-Objekt ist wie folgt aufgebaut und um das Verständnis des Aufbaus zu erleichtern, sind die entsprechenden Zeilennummern des nachfolgenden Listings in Klammern notiert.

1. An erster Stelle steht das JSON Objekt, das die Objekte *booklist* (Zeile 1) und *bookindex* (Zeile 14) enthält. Die *booklist* wiederum enthält einen Array vom Typ textitbooklist (Zeile 2).
2. In dem Array befinden sich Objekte von 1..n, wobei n der Anzahl gefundener Bucheinträge entspricht.

   In einem Eintrag des Arrays sind folgende Informationen gespeichert:

   a) Nach wem oder was gesucht wurde ist in *search* (Zeile 4) gespeichert.

   b) Der Titel des Artikels wird in *title* (Zeile 5) gespeichert.

   c) Der Autor/ die Autoren werden in *authors* (Zeile 6) gespeichert.

   d) Die Beschreibung des Artikels wird in *description* (Zeile 7) gespeichert.

   e) Die URL zu dem Artikel wird in *link* (Zeile 8) gespeichert.

   f) Die Anzahl, wie oft der Artikel zitiert wurde, wird in *cite* (Zeile 9) gespeichert.

   g) Wo und von wem der Artikel publiziert wurde, wird in *puplication* (Zeile 10) gespeichert.
3. In dem Objekt *bookindex* sind folgende Informationen gespeichert:

   a) Nach wem oder was gesucht wurde in *search* (Zeile 15).

   b) Wann danach gesucht wurde in *data* (Zeile 16).

   c) Die Anzahl aller Zitate aufsummiert in *totalcitation* (Zeile 17).

   d) Der berechnete H-Index in *hIndex* (Zeile 18).

   e) Der berechnete G-Index in *gIndex* (Zeile 19).

In Listing 4 ist der Aufbau des JSON-Objekts veranschaulicht.

```
{"booklist":
  {"booklist":[
        {
          "search":"manfred+reichert",
          "title":"Dynamische Ablaufänderungen in Workflow-Management-
              Systemen",
          "authors":"M Reichert ",
          "description":"Workflow-Management-Syst(WfMS) unterstützen die
               elektronische Abwicklung von Arbeitsprozessen (engl.
              Workflow). Heutige WfMS erzwingen meist die starre Ausfü
              hrung des vormodellierten Workflow, dh von dem einmal
              geplanten Ablauf kann zur... ''
          "link":"http://dbis.eprints.uni-ulm.de/433/",
          "cites":102,
          "publication":" 2000 dbis.eprints.uni"},
        }
      ]
  }
  "bookindex":{
        "search":"manfred+reichert",
        "date":"22.10.2013",
        "totalCitation":19148,
        "hIndex":65,
        "gIndex":125}}
```

Listing 4: Aufbau der JSON Datei

In dieser Arbeit wird die Interaktion mit dem Web Service am Beispiel eines mobilen Gerätes von (z.B. iPad, iPhone, etc.) bzw. einer dafür entwickelten nativen App gezeigt [14]. Alle hier dargestellten Bilder der Anwendung wurden in einem iPhone retina (3,5 inch) Simulator mit iOS 7.0 erstellt.
Die Navigation innerhalb der App ist durch einen Tab Bar Controller realisiert (siehe Abbildung 15).

Abbildung 15: Navigation der Application

Dieser besteht aus den Tab Bar Items *Search*, *History* und *More*, mit welchen es möglich ist, sich durch die App zu navigieren.

Wird die App gestartet, befindet man sich im View Controller *Search* der App. Diese realisiert die **FA#1: Einfache Suche** der App. Wie schon im Kapitel *Grundlagen* vorgestellt, ist dort nur die Suche nach einem Parameter möglich (siehe Abbildung 16 links). Durch Einblenden der erweiterten Suchmöglichkeiten wird **FA#2: Erweiterte Suche** der App realisiert. Dort kann nach allen Parametern gesucht werden, wie Sie im Kapitel *Grundlagen* vorgestellt wurden (siehe Abbildung 16 rechts).

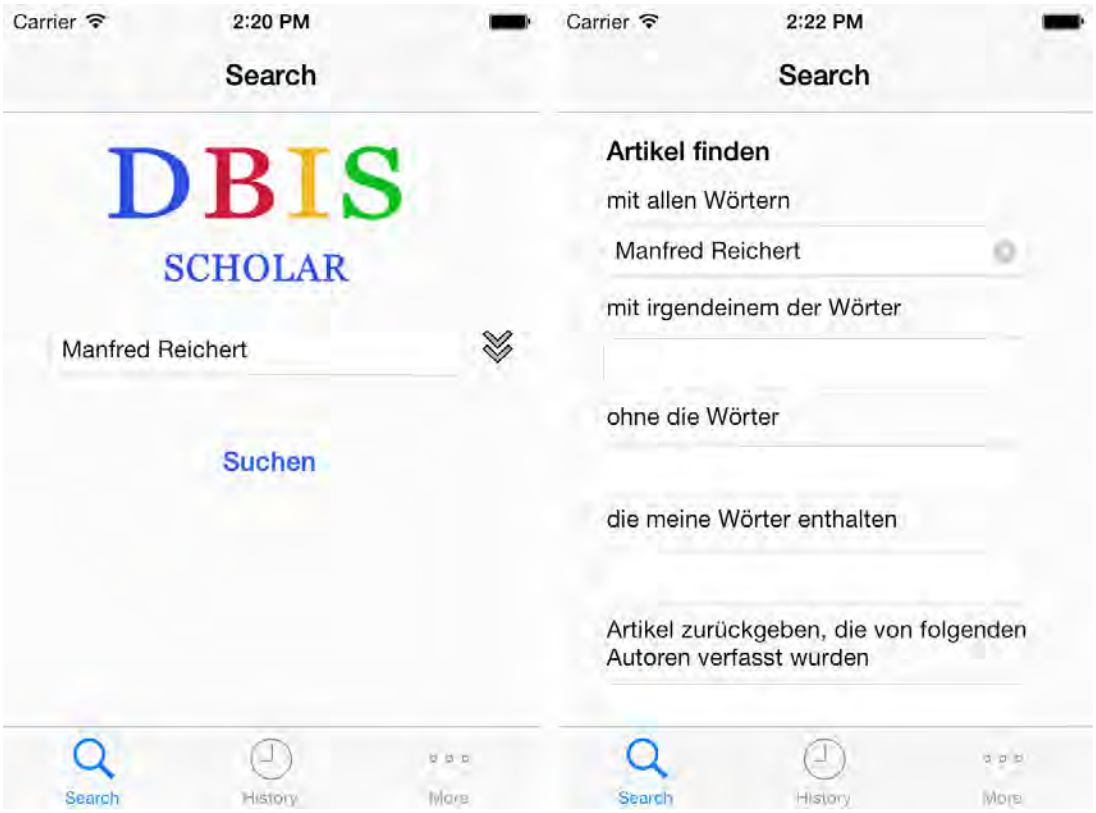

Abbildung 16: Search View

Eine einfache oder erweiterte Suchanfrage wird per NSURLRequest an den Web Service geschickt. Dieser antwortet auf die Anfrage mit einer JSON-Objekt oder einer Fehlermeldung. Die JSON-Objekt wird in einem NSDictionary gespeichert. Das NSDictionary wird danach nach den verschiedenen Informationen (Titel, Autor, Zitate, Beschreibung, Webadresse, Publiziert) mit der Methode *[diction objectForKey:@"..."]* zerlegt und separat in einem Objekt vom Typ *bookentry* in einem NSMutableArray gespeichert (siehe Listing 5).

```
1  NSDictionary *allwebdataDictionary=[NSJSONSerialization
       JSONObjectWithData:webData options:0 error:nil];
2
3      NSDictionary *books= [allwebdataDictionary objectForKey:@"booklist
           "];
4      NSArray *arrayOfBooklist = [books objectForKey:@"booklist"];
5
6      for (NSDictionary * diction in arrayOfBooklist) {
7          NSString *title= [diction objectForKey:@"title"];
8          NSString *authors= [diction objectForKey:@"authors"];
9          NSString *cites= [diction objectForKey:@"cites"];
10         NSString *description= [diction objectForKey:@"description"];
11         NSString *link= [diction objectForKey:@"link"];
12         NSString *publication =[diction objectForKey:@"publication"];
13
14         bookEntry =[Bookentry new];
15         bookEntry.atitle=title;
16         bookEntry.authors=authors;
17         bookEntry.cites=[NSString stringWithFormat:@"%@", cites];
18         bookEntry.description=description;
19         bookEntry.link=link;
20         bookEntry.publication=publication;
21
22         [booklist addObject:bookEntry];
23     }
24     NSDictionary *bookindex = [allwebdataDictionary objectForKey:@"
           bookindex"];
25     NSString *searchfield= [bookindex objectForKey:@"search"];
26     NSString *gindex = [bookindex objectForKey:@"gIndex"];
27     NSString *hindex = [bookindex objectForKey:@"hIndex"];
28     NSString *totalciation = [bookindex objectForKey:@"totalCitation"];
29
30     bookIndex = [Bookindex new];
31
32     bookIndex.search=searchfield;
33     bookIndex.gIndex=gindex;
34     bookIndex.totalCitation=totalciation;
35     bookIndex.Hindex=hindex;
```

Listing 5: Speichern der Informationen

Jetzt stehen die Daten in dem NSMutableArray *booklist* für die Anzeige in der App zur Verfügung. Damit wird die **FA#4: Anzeigen der Datenliste** der App realisiert. Die Informationen werden in einer Table View angezeigt.

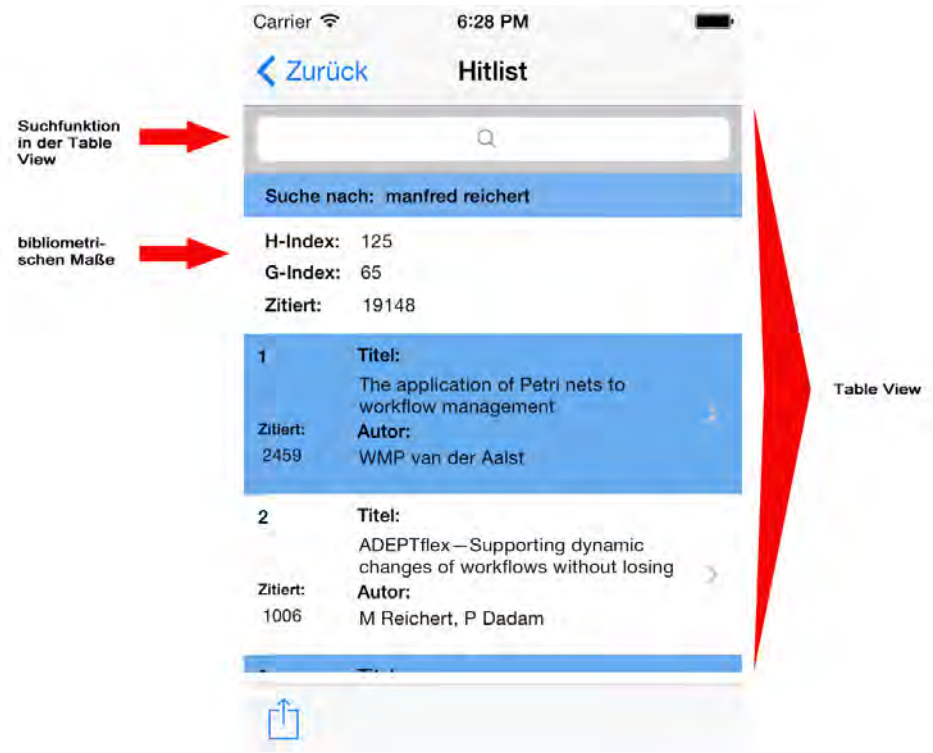

Abbildung 17: Tabelle der Trefferliste

Im ersten Eintrag der Tabelle werden der *Suchbegriff*, der *H-Index*, der *G-Index* und die *Gesamtzahl aller zitierten Werke* angezeigt (siehe Abbildung 17).

Dieses realisiert die **FA#10: Anzeigen der berechneten bibliometrischen Maße** der App.
Über die nachfolgenden Treffer in der Tabelle werden nur die wichtigsten Informationen angezeigt.

1. Titel
2. Autor(en)
3. Anzahl der Zitate

Dadurch wird ein einfacher und überschaubarer Aufbau erhalten. Zudem sind die Zeilen der Tabelle unterschiedlich eingefärbt und durchnummeriert, um die Navigation in der Tabelle zu vereinfachen (siehe Abbildung 18).
Zudem kann in der Tabelle nach Titel und Autoren durch einen Suchfunktion durchsucht werden, dieses realisiert **FA#11: Suche innerhalb der Datenliste** der App.

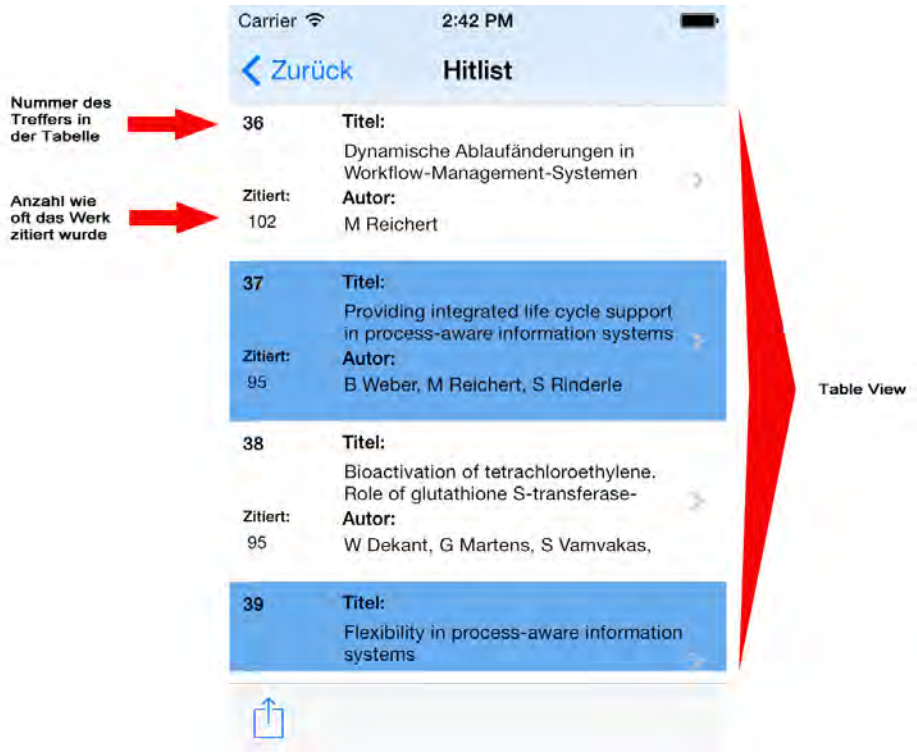

Abbildung 18: Tabelle der Trefferliste

Sollte es vorkommen, dass eine Information nicht vorliegt, oder vom Parser nicht erkannt wurde, wird dieses durch *Notset* angezeigt (siehe Abbildung 19).

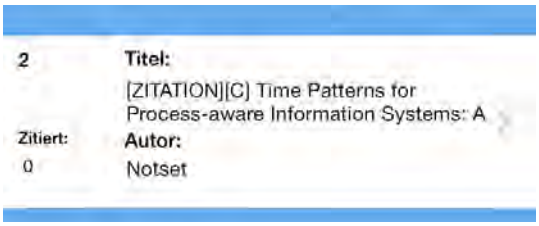

Abbildung 19: Treffer mit Notset

Um die **FA#5: Detaillierte Anzeige eines Treffers** der App zu realisieren, werden die Daten an den nächsten View Controller weitergegeben. Dort werden die Daten um die Informationen *Beschreibung* und *Publiziert* erweitert und ebenfalls als Tabelle vom Typ Table View angezeigt (siehe Abbildung 26). Die Überschriften sind zur besseren Übersichtlichkeit blau eingefärbt.

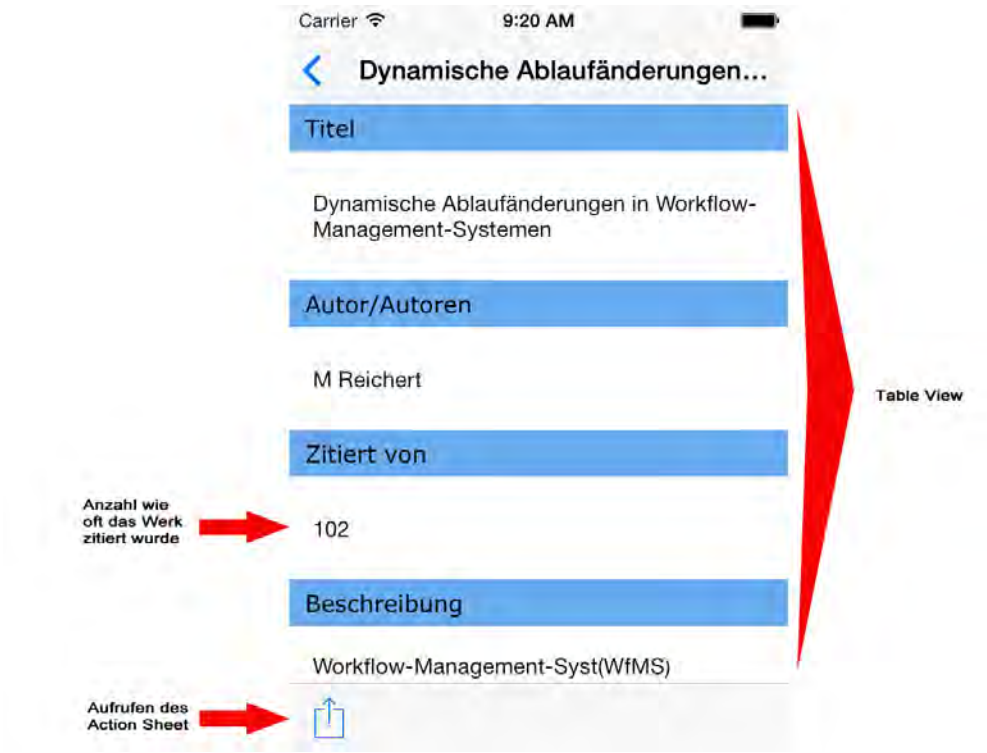

Abbildung 20: Detailansicht eines Treffers

Von diesem Punkt aus ist es möglich, sich die hinterlegte Webseite eines Treffers in einer UIWebView anzeigen zu lassen, was die **FA#3: Anzeigen der Webseite des Treffers** der App realisiert.
Zudem ist es möglich, ein UIAction Sheet aufzurufen. Es realisiert die funktionalen Anforderungen der App:

- **FA#6: Drucken der Daten**
- **FA#7: Versenden der Daten**
- **FA#8: Kopieren der Daten/Weblinks**

Je nachdem von welcher View das UIAction Sheet aufgerufen wird, stehen unterschiedliche Funktionen zur Verfügung. Wird es von *Hitlist* aufgerufen, werden nur die zwei Funktionen *Drucken* und *E-Mail*, der *Hitlist* angeboten. Wird es hingegen von der *detaillierten Ansicht* aufgerufen, stehen zudem *Message und Copy* zur Verfügung (siehe Abbildung 21). Bei einem Aufruf von der Website steht zudem *Add to Reading List* zur Verfügung, dafür fällt *Drucken* weg.

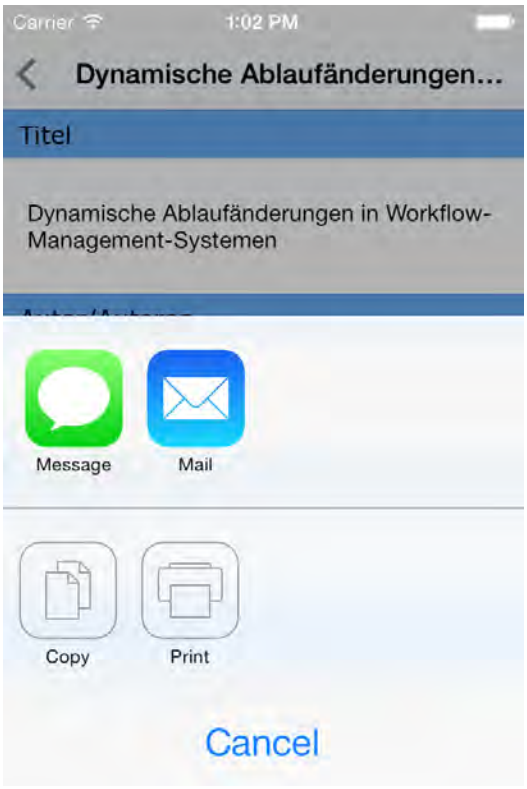

Abbildung 21: Action Sheet

Sollen die Daten gedruckt oder kopiert werden, wird falls der Aufruf aus der *Hitlist* kommt, zuerst abgefragt, wie viele Treffer verwendet/angezeigt werden sollen (siehe Abbildung 22).

Abbildung 22: Anzahl der zu verwendenden Treffer

Ansonsten fällt diese Anfrage weg. In beiden Fällen wird ein PDF-Dokument generiert, das die Daten, die auch in der View angezeigt werden, beinhaltet (siehe Abbildung 23).

Dynamische Ablaufänderungen in Workflow-Management-Systemen

Autor/Autoren:	M Reichert
Zitiert von:	102
Beschreibung:	Workflow-Management-Syst(WfMS) unterstützen die elektronische Abwicklung von Arbeitsprozessen (engl. Workflow). Heutige WfMS erzwingen meist die starre Ausführung des vormodellierten Workflow, dh von dem einmal geplanten Ablauf kann zur ...
Publiziert:	2000 dbis.eprints.uni
Link:	http://dbis.eprints.uni-ulm.de/433/

Abbildung 23: PDF der Detailansicht eines Treffers

Die **FA#3: Speicherung der Parameter der vergangenen Suchen** der App wird durch eine UITable View realisiert. Die Tabelle ist in der Lage bis zu 10 vergangene Suchen, sortiert nach ihrer Aktualität, anzuzeigen. Zudem wird das Datum der Speicherung, und ob es sich um eine einfache oder erweiterte Suche handelt, mitangezeigt (siehe Abbildung 24). Die Daten sind über die App-Laufzeit hinaus in einer plist gespeichert und können so bei jedem App-Neustart wieder angezeigt werden und gehen nicht verloren.

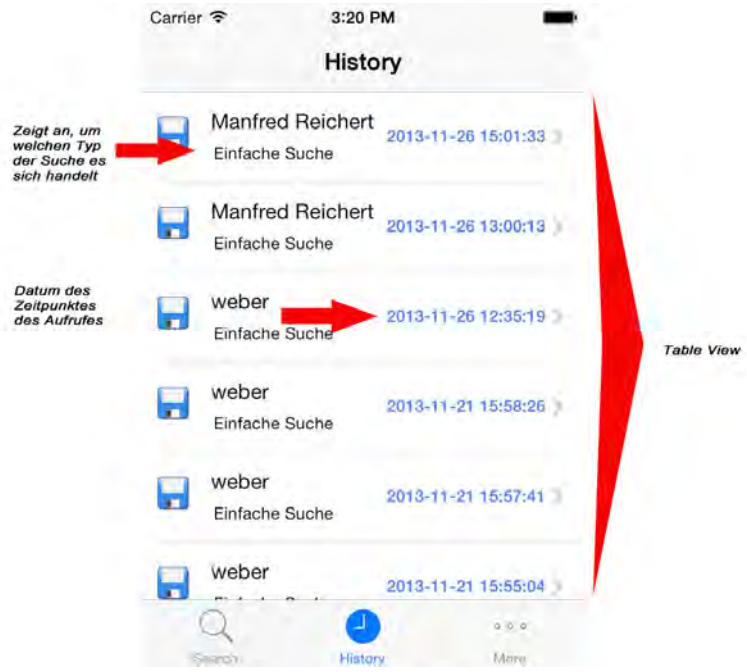

Abbildung 24: Vergangene Suchen

Nachdem nun das gesamte System inklusive seiner Komponenten vorgestellt wurde, wird nun im nachfolgenden Kapitel überprüft, inwieweit die Anforderungen, die in Kapitel 3 an den Web Service und die App gestellt wurden, erfüllt worden sind.

5. Anforderungsabgleich

Im Kapitel *Anforderungsanalyse* wurden die funktionalen und nichtfunktionalen Anforderungen an das System vorgestellt. Nun soll in einer Tabelle abgeglichen werden, inwieweit diese Anforderungen erfüllt wurden oder nicht.

5.1. Web Service

In diesem Unterkapitel wird nun deshalb untersucht, ob die Anforderungen an den Web Service erfüllt wurden oder nicht.

Funktionale Anforderungen

Nr.	Name	Erfüllt	Erklärung
1	Einfache Suche bei Google Scholar	√	Die einfache Suche wird vom Web Service angeboten und unterstützt.
2	Erweiterte Suche bei Google Scholar	√	Die erweiterte Suche wird vom Web Service angeboten und unterstützt.
3	Parsen der Daten	√	Die Daten werden vom Web Service geparst und dann per JSON zur Verfügung gestellt.
4	Berechnung des H-Index	√	Der Web Service berechnet den H-Index, wie in Kapitel 2.2 vorgestellt.
5	Berechnung des G-Index	√	Der Web Service berechnet den G-Index, wie in Kapitel 2.3 vorgestellt.
6	Speicherung der Daten in einer Datenbank	√	Die Daten werden in einer Mysql Datenbank persistent gespeichert und stehen dadurch für erneute Anfragen zur Verfügung.
7	Berechnung der Zeitdifferenz des letzten Aufrufs	√	Bei jeder Speicherung eines Treffers wird ein Zeitstempel mit gespeichert, so kann die Differenz zum aktuellen Datum berechnet werden.
8	Löschen nicht mehr benötigter Daten		Die Methode wird zwar in der Datenbank angeboten, aber nicht selbstständig ausgeführt.

Nr.	Name	Erfüllt	Erklärung
9	Umgehung der Limitation der Zugriffe auf Google Scholar	√	Der Web Service stellt die Anfragen an Google Scholar über verschiedene Proxyserver, wodurch seine IP-Adresse nicht immer dieselbe ist.
10	Intelligente Wahl des Proxyservers	√	Wie in Kapitel 4.2 vorgestellt, wird ein Proxyserver zufällig und nach seiner Qualität ausgewählt.
11	Bewertung der Proxyserver	√	Die Proxyserver werden anhand ihrer Geschwindigkeit bewertet.
12	Löschen von Proxyservern	√	Proxyserver, die mehrmals zu langsam waren, werden automatisch gelöscht.

Nichtfunktionale Anforderungen

Nr.	Name	Erfüllt	Erklärung
1	Schnelle Reaktion auf Anfragen	√	Durch cashing und die REST Architektur.
2	Wartbarkeit der Datenbank	√	Ist durch die aussagekräftige Benennung der Attribute in der Datenbank erfüllt.
3	Dauer des Daten-Cachings	√	Ist durch den gut kommentierten Code erfüllt. Dadurch sind entsprechende Stellen, die geändert werden müssen, leicht zu finden.

5.2. App

Ob die Anforderungen an die App erfüllt wurden oder nicht, wird in diesem Unterkapitel untersucht.

Funktionale Anforderungen

Nr.	Name	Erfüllt	Erklärung
1	Einfache Suche	√	Die App realisiert die einfache Suche.
2	Erweiterte Suche	√	Die App realisiert die erweiterte Suche.
3	Speicherung der Parameter vergangener Suchen	√	Die App verwaltet eine Tabelle mit bis zu 10 vergangenen Suchbegriffen.
4	Anzeige der Trefferliste	√	Die Daten werden in einer Tabelle in der App angezeigt.
5	Detaillierte Anzeige eines Treffers	√	Durch Anklicken eines Treffers gelangt man zu einer detaillierten Ansicht.
6	Drucken der Daten	√	Daten können über Airprint gedruckt werden.
7	Versenden der Daten	√	Die Daten können als ein PDF Dokument generiert und so per Mail versendet werden.
8	Kopieren der Daten/Weblinks	√	Das Kopieren der Daten und Weblinks wird im „Actionsheet“ angeboten und steht so funktionell zur Verfügung.
9	Anzeigen der Website des Treffers	√	Durch anklicken einer URL in der Trefferliste wird man automatisch auf die entsprechende Website weitergeleitet.
10	Anzeigen der berechneten bibliometrischen Maße	√	Die Daten werden im ersten Eintrag der Ergebnistabelle in der App angezeigt.
11	Suche innerhalb der Datenliste	√	Da eine Suchfunktion in der Table View angeboten wird.

Nichtfunktionale Anforderungen

Nr.	Name	Erfüllt	Erklärung.
1	Einfache Navigation in der App	√	Erfüllt, durch weglassen aller unwichtigen Elemente, die Verwendung von einheitlichen Farben und der von Apple bereitgestellten Navigations-Elemente.
2	Anzeigen von Aktivitäten der App	√	Alle Aktivitäten, wie zum Beispiel die einfache Suche in der App, werden durch ein *UIActivityIndicator* angezeigt.
3	Die App soll ab iOS 6 universell laufen	√	Die App ist ab iOS auf allen mobilen Geräten von Apple lauffähig.

6. Zusammenfassung

Das erreichte Ergebnis der Arbeit, die Konzeption und Entwicklung einer plattformunabhängigen, intelligenten Middleware und der zugehörigen lauffähigen App, wird im Unterkapitel *Fazit* formuliert. Ebenfalls wird eine Begründung für das verwendete Konzept gegeben und Probleme geschildert. Im Ausblick wird ein Algorithmus vorgestellt, der die Geschwindigkeit des parsens der Daten von Google Scholar erhöht. Dies führt zu einer schnelleren Verfügbarkeit der Treffer. Zudem wird vorgestellt, wie der Web Service für die Entwicklung von Apps anderer Software-Plattformen verwendet werden kann.

6.1. Fazit

Das Ziel dieser Arbeit, eine plattformunabhängige Middleware zu konzeptionieren und implementieren, wurde durch die Implementierung eines RESTful Web Services erreicht. Die im Rahmen dieser Arbeit entwickelte iOS App ist für die Geräte iPhone, iPad und iPod von Apple unter iOS 6.0 bis iOS 7.1 kompatibel. Um die erreichten Ergebnisse zu evaluieren, wurde im Rahmen des System Requirement Engineering anhand einer Anforderungsanalyse (Kapitel 3) ein Anforderungsabgleich (Kapitel 5) durchgeführt. Der Anforderungsabgleich wurde für den Web Service und die App getrennt durchgeführt. Bezüglich der App wurden alle funktionalen und nichtfunktionalen Anforderungen erfüllt. Die nichtfunktionalen Anforderungen an den Web Service wurden ebenfalls vollständig umgesetzt. Die **FA#8 Löschen nicht mehr benötigter Daten** des Web Service wurde im Rahmen dieser Arbeit nicht verwirklicht. Die Methode wird zwar in der Datenbank angeboten, aber nicht selbstständig ausgeführt. Sobald der Web Service auf den Servern der Universität Ulm integriert ist, kann dies behoben werden, indem ein Skript in Form eines *Cron Jobs* geschrieben wird, das die Methode zum löschen der alten und nicht mehr benötigten Datensätze aufruft.
Aufgrund der Fragmentierung mobiler Plattformen (iOS, Android uvm.) und der Tatsache, dass jede Plattform ihre eigene angepasste App benötigt, ist es sinnvoll, einen plattformunabhängigen Web Service zu verwenden. Dieser übernimmt die zentralen Aufgaben (parsen, cachen und die Umgehung der Limitation an Zugriffe auf Google Scholar durch Proxyserver) für jedes beliebige Betriebssystem. Dadurch ist die Entwicklung der App auf einem anderen Betriebssystem erleichtert, da die zentralen Aufgaben nicht nochmals implementiert werden müssen. Es wurde ein RESTful Web Service verwendet, da er direkt über ein URI angesprochen werden kann und durch den Einsatz von HTTP zahlreiche Vorteile gegenüber SOAP und XML-RPC bietet. Diese sind seine hohe Perfomance, seine leichte Skalierbarkeit, eine hohe Verfügbarkeit und einfache Erweiterbarkeit [12].

6.2. Ausblick

Im Ausblick auf die Zukunft könnte der im Rahmen dieser Arbeit erstellte RESTful Web Service dem Streben nach immer mehr Schnelligkeit nicht gerecht werden. Da er zunächst Seite 1 parst, dann die zweite Seite aufruft und diese parst. Das Aufrufen der folgenden Seiten wird so lange ausgeführt, bis alle Daten der Seiten geparst wurden, oder insgesamt 1000 Eintrage geparst vorliegen. Dies nimmt viel Zeit in Anspruch, da Seite für Seite aufgerufen wird. Würde der Web Service mehrere Seiten parallel aufrufen und parsen, wäre er je nachdem wie viel er parallel ausführt, proportional schneller.

Ein denkbarer Algorithmus sieht folgendermaßen aus:

1. Beim ersten Aufruf werden, wie beim jetzigen System, die ersten 10 Seiten in einer Liste gespeichert.

2. Diese werden jedoch nicht nacheinander aufgerufen, bis keine Seite mehr vorhanden ist, sondern es werden mehrere separate Aufrufe gestartet. Dazu wird immer die 10. Seite aufgerufen, welche dann wiederum für die nächsten 9 Seiten zuständig ist.

3. Damit kann in Abhängigkeit der Anzahl an Treffern eine fünffache Geschwindigkeitssteigerung erreicht werden.

Der Algorithmus bringt einige Problem mit sich, da zu keiner Zeit bekannt ist, wie viele Trefferseiten vorliegen, außer man befindet sich auf den letzen 5 Seiten oder die Suche liefert nur so wenige Treffer, dass nur wenige Seiten vorliegen. Der Web Service müsste also auf Verdacht die Seiten aufrufen und damit umgehen können, dass viele Aufrufe ins Leere laufen und eventuell gar nichts zurückgegeben wird. Zudem müsste dafür gesorgt werden, dass alle Daten der Aufrufe wieder zusammengeführt werden. Die Erstellung des Algorithmus hätte den Rahmen dieser Arbeit gesprengt.

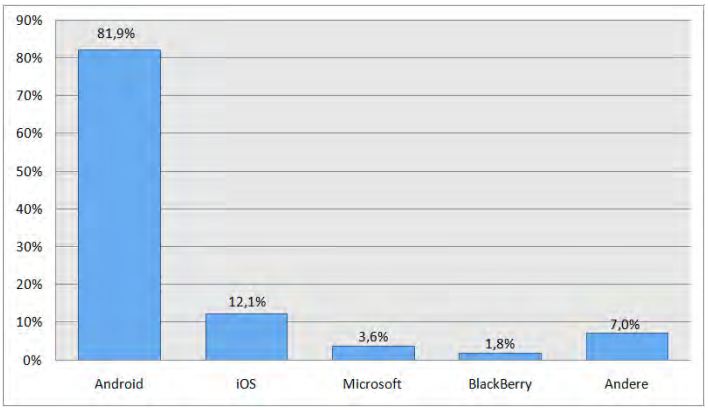

Abbildung 25: Marktanteil verkaufter Smartphones nach Betriebssystemen in % [6]

Nach dem Smartphone Betriebssystem Markt im 3. Quartal 2013 (siehe Abbildung 25) ergibt sich, dass Smartphones mit Android-Betriebssystem die meist verkauften sind. Sie haben einen Marktanteil von 81,9%. Dabei machen Geräte mit iOS-Betriebssystem mit 12,1% nicht einmal ein Achtel des Markts aus. Die anderen Betriebssysteme (Mircosoft, BlackBerry etc.) machen jeweils weniger als 4% aus.
Da jeder Hersteller mit seinem Betriebssystem attraktiv bleiben will, ist es vorteilhaft für Firmen, die Apps entwickeln, mehrere Software-Plattformen zu unterstützen. Aufgrund des hier gewählten Architektur-Konzepts ist es problemlos möglich, eine Android App für Google Scholar zu entwickeln, die den implementierten RESTful Web Service als Middleware verwendet.
Mit Hilfe einer hybriden Anwendung, welche unter Verwendung eines Cross-Development-Frameworks entwickelt wird, könnten weitere mobile Plattformen bedient werden, wobei dieser Anwendungstyp deutliche Nachteile in Sachen Performance und Sensor-Zugriff besitzt [7].

7. Quellenverzeichnis

[1] M. Beutel, *Evaluation von XML und Java*. GRIN Verlag, 2003.

[2] ECMA-262 (ISO/IEC 16262), 3 rd Ed., ECMAScript® Language Specification. URL: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf, zuletzt besucht: 01.11.2013.

[3] ECMA-404. The JSON Data Interchange Format, 1 st Ed., ECMA International. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, zuletzt besucht: 01.11.2013.

[4] L. Egghe, "Theory and practise of the g-index." Akadémiai Kiadó, co-published with Springer Science+ Business Media BV, Formerly Kluwer Academic Publishers BV, 2006, vol. 69, no. 1, pp. 131–152.

[5] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[6] Gartner, worldwide Smartphone Sales to End Users by Vendor in 3Q13. URL: http://www.gartner.com/newsroom/id/2623415, zuletzt besucht: 11.12.2013.

[7] P. Geiger, R. Pryss, M. Schickler, and M. Reichert, "Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices," no. UIB-2013-09. University of Ulm, October 2013, Technical Report, URL: http://dbis.eprints.uni-ulm.de/972/, zuletzt besucht: 09.12.2013.

[8] Google Scholar. URL: http://scholar.google.de/intl/de/scholar/about.html, zuletzt besucht: 03.09.2013.

[9] J. E. Hirsch, "An index to quantify an individual's scientific research output," vol. 102, no. 46. Proceedings of the National academy of Sciences of the United States of America, 2005, pp. 16 569–16 572.

[10] Jericho HTML Parser. URL: http://jericho.htmlparser.net/docs/index.html, zuletzt besucht: 21.11.2013.

[11] P. Mayer, "Google Scholar als akademische Suchmaschine," in *VÖB-Mitteilungen*, 2009, vol. 62, pp. 18–28.

[12] I. Melzer, S. Werner, P. Sauter, A. H. von Thile, M. Flehmig, B. Zengler, W. Dostal, P. Tröger, B. Stumm, M. Lipp, and M. Jeckle, *Service-orientierte Architekturen mit*

Web Services: Konzepte-Standards-Praxis. Elsevier, Spektrum, Akad. Verlag, 2007, no. 2.

[13] A. Robecke, R. Pryss, and M. Reichert, "Development of an iPhone business application," in *CAiSE Forum-2011*, ser. Proceedings of the CAiSE'11 Forum at the 23rd International Conference on Advanced Information Systems Engineering, vol. 73. CEUR Workshop Proceedings, June 2011, URL: http://dbis.eprints.uni-ulm.de/941/, zuletzt besucht: 09.12.2013.

[14] J. Schobel, M. Schickler, R. Pryss, H. Nienhaus, and M. Reichert, "Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned," in *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, May 2013, pp. 509–518, URL: http://dbis.eprints.uni-ulm.de/918/, zuletzt besucht: 09.12.2013.

[15] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007. URL: http://www.w3.org/TR/soap12-part1/, zuletzt besucht: 01.11.2013.

[16] L. Sommerville, *Software Engineering.* Pearson Studium, 2012, vol. 9. aktualisierte Auflage.

[17] C. Ullenboom, *Java 7 - Mehr als eine Insel, das Expertenbuch zu den Java SE Bibliotheken.* Galileo Computing, 2012, vol. 1.

[18] S. Zauchner, *Zur metrischen Evaluierung von Publikationsleistungen.* Donau-Universität Krems, 2010.

8. Abbildungsverzeichnis

9. Tabellenverzeichnis

10. Listing-Verzeichnis

Anhang

A. Storyboard

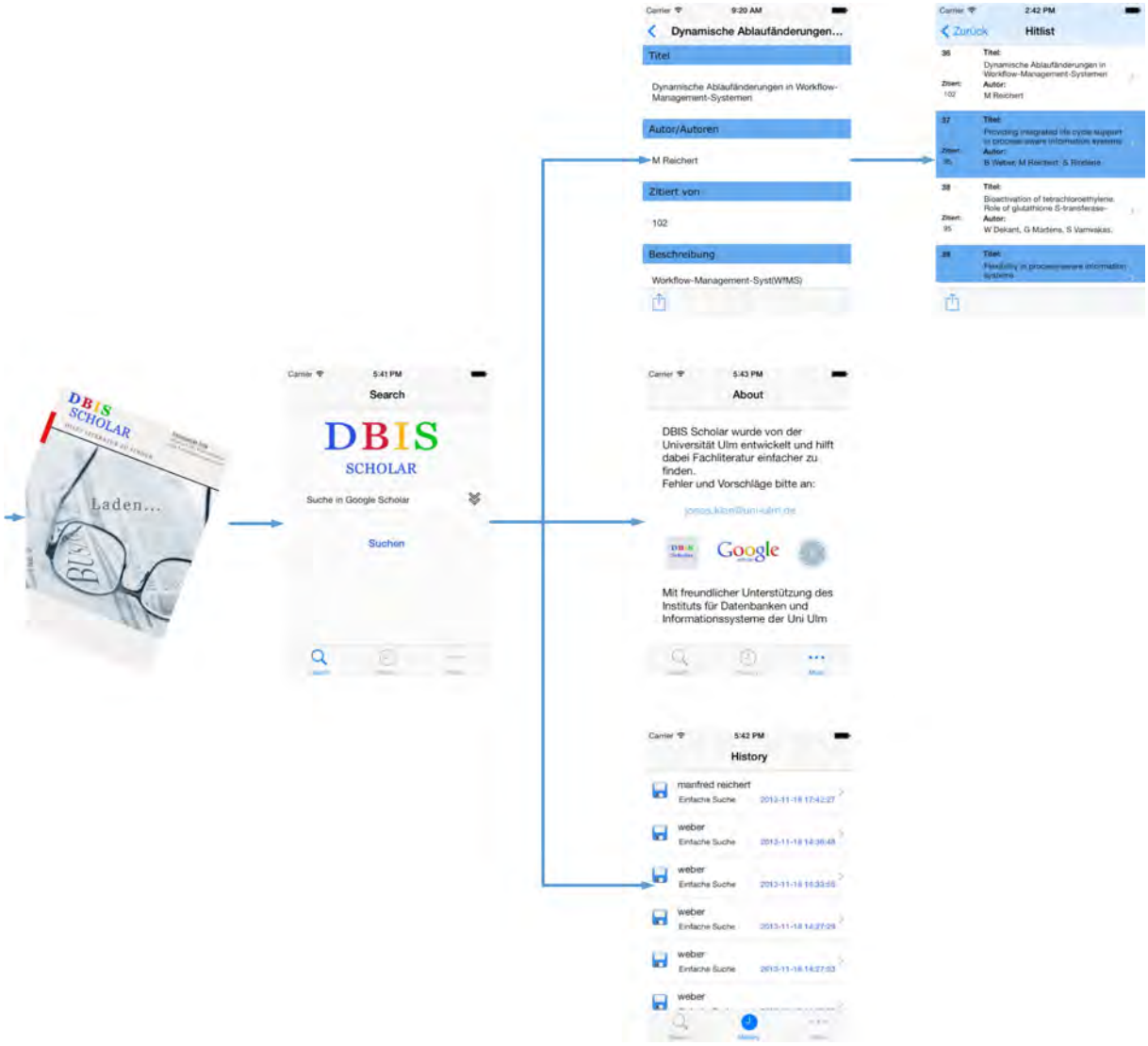

Abbildung 26: Storyboard der App

B. Quellcode

Auf der beiliegenden CD befindet sich der Quellcode des entwickelten Web Service, sowie der App. Der Web Service wurde in Eclipse, in einer Java Entwicklungsumgebung entwickelt. Die dazugehörige iOS 7 App wurde in Xcode 5.0 erstellt.

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)