JUST DO
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

DezSys07
SOA and RESTful Webservice

Dezentrale Systeme
5BHIT 2015/16

Geyer Stefan, Ritter Mathias

Version 1.0

Betreuer: Prof. Micheler
Note:

Begonnen am 4. Dezember 2015
Beendet am 17. Dezmber 2015

Inhaltsverzeichnis

1 Aufgabenstellung

Das neu eröffnete Unternehmen **iKnow Systems** ist spezialisiert auf **Knowledgemanagement** und bietet seinen Kunden die Möglichkeiten Daten und Informationen jeglicher Art in eine Wissensbasis einzupflegen und anschließend in der zentralen Wissensbasis nach Informationen zu suchen (ähnlich wikipedia).
Folgendes ist im Rahmen der Aufgabenstellung verlangt:

- Entwerfen Sie ein Datenmodell, um die Eintraege der Wissensbasis zu speichern und um ein optimitiertes Suchen von Eintraegen zu gewaehrleisten. **[2Pkt]**

- Entwickeln Sie mittels RESTful Webservices eine Schnittstelle, um die Wissensbasis zu verwalten. Es muessen folgende Operationen angeboten werden:
  - **Hinzufuegen** eines neuen Eintrags
  - **Aendern** eines bestehenden Eintrags
  - **Loeschen** eines bestehenden Eintrags

- Alle Operationen muessen ein Ergebnis der Operation zurueckliefern. **[3Pkt]**

- Entwickeln Sie in **Java** ein **SOA Webservice**, dass die Funktionalitaet **Suchen** anbietet und das **SOAP** Protokoll einbindet. Erzeugen Sie fuer dieses Webservice auch eine **WSDL**-Datei. **[3Pkt]**

- Entwerfen Sie eine **Weboberflaeche**, um die **RESTful Webservices** zu verwenden. **[3Pkt]**

- Implementieren Sie einen **einfachen Client** mit einem User Interface (auch Commandline UI moeglich), der das **SOA Webservice** aufruft. **[2Pkt]**

- Dokumentieren Sie im weiteren Verlauf den Datentransfer mit SOAP. **[1Pkt]**
- Protokoll ist erforderlich! **[2Pkt]**

Info:
Gruppengroesse: 2 Mitglieder
Punkte: 16
Zum Testen bereiten Sie eine Routine vor, um die Wissensbasis mit einer **1 Million Datensaetze** zu fuellen. Die Datensaetze sollen mindestens eine Laenge beim Suchbegriff von 10 Zeichen und bei der Beschreibung von 100 Zeichen haben! **Ist die Performance bei der Suche noch gegeben?**

Links:
JEE Webservices:
http://docs.oracle.com/javaee/6/tutorial/doc/gijti.html
Apache Web Services Project:
http://ws.apache.org/

Apache Axis/Axis2:
http://axis.apache.org
http://axis.apache.org/axis2/java/core/

IBM Article: Java Web services - JAXB and JAX-WS in Axis2:
http://www.ibm.com/developerworks/java/library/j-jws8/index.html

2 Ergebnisse

2.1 Datenmodell & Generierung der Testdaten

Als Datenmodell wurde ein JSON-Objekt definiert, welches ein Attribut `name` und ein Attribut `description` enthält. Diese werden in einer dokumentenorientierten NoSQL Datenbank, MongoDB, abgelegt.

Um eine Million Testdaten zu generieren, wurde der Datengenerator der Seite "GenerateData" verwendet. [1] Dieser muss allerdings heruntergeladen werden, da online maximal 100 Datensätze generiert werden können.

Nach dem Herunterladen benötigt man einen funktionsfähigen Webserver, PHP & MySQL Setup, um das Installationsscript auszuführen. Im ersten Schritt gibt man die Daten der Datenbank an. Falls noch kein User und keine Datenbank erstellt wurde, muss diese vorher noch erstellt werden. Danach baut das Script eine Datenbankverbindung auf. Nun werden die Testdaten in die Datenbank geladen. Im letzten Schritt gibt man Username & Passwort zum Aufruf der Seite an. Danach kann man den DataGenerator wie bei der Online-Version verwenden.

Um die oben erwähnten Daten zu generieren, gibt man nun bei der ersten Spalte als Bezeichnung "name" ein. Als Datentyp wählt man Name. In die Optionsspalte schreibt man "Name Name Surname". In der zweiten Spalte gibt man als Bezeichnung "description" ein und wählt "random number of words" aus. In der Optionsspalte stellt man zwischen 20 und 30 Wörter ein. Zuletzt stellt man 100.000 Datensätze (Maximum) und JSON (Simple) ein. Nun klickt man auf Generate und nach ein paar Sekunden lädt das generierte JSON-File herunter. Diesen Vorgang wiederholt man 10 Mal und fügt danach alle Daten zu einer JSON-Datei zusammen.

Nach dem Einlesen der Datensätze (siehe Kapitel 2.6.2) wird zusätzlich eine ID zu jedem Eintrag hinzugefügt, da MongoDB ein eindeutiges Identifikationsmerkmal eines Eintrags benötigt.

Ein Datensatz sieht nach dem Einfügen beispielsweise folgendermaßen aus:

```
{
	"id": "5672e37578bc38ac63f41a38",
	"name": "Iris Carly Schroeder",
	"description": "dolor. Fusce mi lorem, vehicula et, rutrum eu,
		ultrices sit amet, risus. Donec nibh enim, gravida sit amet,
		dapibus id, blandit at, nisi. Cum sociis"
}
```

2.2 Rest-Schnittstelle

URL	Methode	Zweck
`/datarecords`	GET	Rückgabe aller vorhandenen Datarecords (maximal 100)
`/datarecords?name=XYZ`	GET	Rückgabe aller vorhandenen Datarecords, welche im Namen "XYZ" enthalten (maximal 100)
`/datarecords`	POST	Erstellen eines neuen Datarecords. Dazu muss im Body der Eintrag in der Form des in Kapitel 2.1 beschriebenen JSON-Dokuments mitgegeben werden.
`/datarecords/{id}`	GET	Rückgabe des Datarecords mit der in der URL angegebenen ID.
`/datarecords/{id}`	PUT	Update des Datarecords mit der in der URL angegebenen ID. Dazu muss im Body der neue Eintrag in der Form des in Kapitel 2.1 beschriebenen JSON-Dokuments mitgegeben werden.
`/datarecords/{id}`	DELETE	Löschen des Datarecords mit der in der URL angegebenen ID.

Achtung: Als Accept-Header muss immer `application/json` angegeben werden.

Diese Methoden wurden in der Klasse `DataRecordRestController` implementiert. Dabei wird das `MongoDBDataRecordService` verwendet, um auf die Datenbank zuzugreifen. Dieses wird mittels `@Autowired` instanziert.

Für jede Methode wird die URL, die Methode und das Output-Format angegeben, wie man an folgendem Beispiel sehen kann:

```
@RequestMapping(value = "/datarecords", method = RequestMethod.GET, produces = "application/json")
public ResponseEntity<List<DataRecord>> findDataRecordsByName(@RequestParam(value = "name",
defaultValue = "") String name) {
    if (name.length() == 0)
        return new ResponseEntity<>(service.findTop100(), HttpStatus.OK);
    else
        return new ResponseEntity<>(service.findTop100ByNameContainingIgnoreCase(name), HttpStatus.OK);
}
```

In der Methode wird auf den GET-Parameter mit der Annotation `@RequestParam` zugegriffen. Falls kein GET-Parameter angegeben wurde, werden die ersten 100 Einträge zurückgegeben. Falls ein GET-Parameter angegeben wurde, werden die ersten 100 Einträge zurückgegeben, die den im GET-Parameter angegebenen Namen enthalten. Die Rückgabe erfolgt als `ResponseEntity`, um zusätzlich einen HTTP-Status zurückzugeben.

Bei der Implementierung hat uns vor allem das Spring-Tutorial geholfen [6].

2.3 Suche mittels SOAP

Um Dateneinträge leichter finden zu können, soll mithilfe von SOAP eine Suchfunktion implementiert werden. Auch dazu existiert ein Spring Modul, mit welchem eine SOAP-Anwendung automatisch ausgeführt werden kann. [2]

Konkret müssen zwei Maven-Dependencies hinzugefügt werden: Einerseits das Spring WS Modul an sich und andererseits das WSDL4J Modul, welches später zum Generieren einer WSDL-Definition verwendet wird.
Jedoch arbeitet Spring in Bezug auf WSDL-Definitionen etwas anders: Anstatt ein WSDL-File zu erstellen, wird eine XSD-Schema Datei erzeugt. Darin müssen alle Requests, Responses und Models festgelegt werden, welche später bei der Kommunikation mittels SOAP verwendet werden. Des Weiteren muss ein Namespace definiert werden. Dies geschieht im Schema Tag, sollte eindeutig sein und die Applikation identifizieren.

Danach wird mithilfe des Maven-Plugins „jaxb2“ die Schema-Datei in entsprechende Java Source-Files umgewandelt. Diese enthalten vordefinierte Attribute und Methoden (wie im Schema definiert). Das Plugin läuft standardmäßig bei einem normalen „install“ Befehl von Maven mit. Wird das Plugin mehrmals ausgeführt, werden vorgenommene Änderungen überschrieben.
Falls eine generierte Klasse (z.B. ein Model) bereits existiert, können die Annotationen auch in die andere übernommen werden um so eine Redundanz zu verhindern.
Um nun auf eine Datenbank zugreifen zu können, wird ein Repository benötigt. In diesem Fall wurde das Mongo-Repository weiterverwendet.

Um das Service nun verwenden zu können, müssen noch eine Konfiguration und ein Endpunkt definiert werden. Der Endpunkt verbindet Request und Response an einem konkreten URI in einem festgelegtem Namespace.
Die Konfigurationsklasse erstellt aus dem XSD-Schema eine WSDL 1.1 Definition und passt etwaige Werte an.

2.4 SOAP Client

Damit die SOAP Suche ausprobiert und angewandt werden kann, wurde ein einfacher SOAP-Client als CLI Programm implementiert.

Der Ablauf ist relativ simpel. Zuerst werden die CLI-Argumente eingelesen und aufbereitet. Konkret sind das die Parameter Name (Suchbegriff), Host, Port und Output wobei die letzten drei optional sind.

Mit diesen Daten wird nun ein SOAP-Request erzeugt. Dieser wird daraufhin an den Server gesendet und die entsprechende Nachricht ausgegeben. Wurde ein Output Parameter definiert, wird die Antwort stattdessen in das angegebene File gespeichert, sofern dieses vorhanden ist.

2.5 Datenzugriffsschicht

Es wurde eine Model Klasse mit drei Feldern (ID, Name und Beschreibung) definiert. Des Weiteren wurde das Builder-Pattern verwendet, um ein Model zu erzeugen. Neben der Tatsache, dass Objekte so einfacher erzeugt werden können, werden die angegebenen Daten auch validiert bevor das Objekt tatsächlich erzeugt wird.

Um das Model in einer Spring-Anwendung persistieren zu können, werden sogenannte Repositories benötigt. Ein Repository ist die einzige Verbindung, die zur Datenbank gegeben ist, und wird mit der Annotation `@Repository` versehen. Die Datenschnittstelle wird als Interface definiert und erbt vom Interface `MongoRepository`, da MongoDB als Datenbank verwendet wird. Das Elterninterface verfügt bereits über Methoden wie `save`, `delete`, `findAll` usw. Die Schnittstelle kann später, ohne sie konkret implementieren zu müssen, über weitere Spring Annotationen aufgerufen werden.

Möchte man allerdings eigene Methoden definieren, müssen diese nach einem eigenen Schema angelegt werden. Konkret wird die folgende Methode erstellt:

```
List<DataRecord> findTop100ByNameContainingIgnoreCase(@Param("name") String name);
```

Diese Methode gibt die ersten 100 Einträge zurück, welche den als Parameter übergebenen Namen beinhalten. Dabei wird nicht auf Groß- und Kleinschreibung geachtet.

Weiters wird eine Konfigurationsklasse für die Datenbank erstellt. Darin werden weitere Parameter, wie etwa Datenbankname, Port und Host, konfiguriert. Die Konfigurationsklasse muss mit der `@Configuration` Annotation versehen werden, damit sie von Spring als solche erkannt wird.

2.6 Konfiguration der Vagrant VM

Um das Deployment der Webapplikation und das Hosten von MongoDB einfacher zu gestalten, wurde eine virtuelle Maschine mittels Vagrant konfiguriert.

In der Datei /vagrant/Vagrantfile wurden folgende Grundparameter eingestellt:

- Betriebssystem: `Debian/Jessie64`
- IP: `192.168.10.200`
- Shared Folder, um auf die Testdaten in der VM zugreifen zu können
- Shellscript für Shell-Provisioning unter `shell/default`

Im Shell-Script wurde dann die weitere Installation/Konfiguration festgelegt, welche nun in den folgenden Unterkapital genauer beschrieben wird.

2.6.1 Installation & Einrichten MongoDB

Mongo DB wird installiert, indem ein zusätzliches Repository hinzugefügt wird. Danach muss noch konfiguriert werden, dass MongoDB von Außen aufrufbar ist. Dabei wird in der `/etc/mongod.conf` die IP `0.0.0.0` eingestellt. Um die Änderungen zu Übernehmen, muss MongoDB neu gestartet werden.

2.6.2 Entpacken & Einspielen der Testdaten

Um die Testdaten zu entpacken, wird `7zip` installiert. Zuerst wird die DB, falls vorhanden, gelöscht. Danach wird die Datei entpackt und mittels `mongoimport` in die Datenbank `webappdb` gespielt.

2.6.3 Installation Java & Konfiguration Tomcat

Um die Anwendung später auf die virtuelle Maschine deployen zu können, muss Java installiert und Tomcat konfiguriert werden. Dabei wird zuerst per Oracle Java 8 Installer das Java 8 JDK installiert. Danach wird Tomcat 8 heruntergeladen, entpackt und in das Verzeichnis `/usr/share/tomcat` verschoben. Bevor Tomcat nun gestartet wird, wird noch ein User angelegt, um mittels dieses Users dann deployen zu können.

2.7 Ausführen

Um die Webapplikation und den SOA-Client auszuführen, muss zuerst eine virtuelle Maschine mittels Vagrant aufgesetzt werden. Dazu muss man VirtualBox, Vagrant und Maven herunterladen und installieren. [3, 4, 5]

1. Vom Root-Verzeichnis des Projekts navigiert man in den Ordner `vagrant` und führt in der Konsole `vagrant up` aus.
2. Im Root-Verzeichnis des Projekts führt man in der Konsole `mvn clean` und `mvn install` aus.
3. Vom Root-Verzeichnis des Projekts navigiert man in den Ordner `web-application` und führt `mvn tomcat7:deploy` aus.
4. Vom Root-Verzeichnis des Projekts navigiert man in `saop-client/final` und führt die in diesem Ordner enthaltene `.jar`-Datei in der Konsole mittels `java -jar <DATEINAMEN> -h 192.168.10.200 -n <SUCHBEGRIFF>` aus.
    a. Ein Output-File kann mittels `-o <DATEINAME>` angegeben werden
    b. Der Port kann mittels `-p <PORTNR>` angegeben werden

Bitte beachten: Bei der Suche und der Ansicht auf der Webseite (bzw. allgemein beim Aufruf der REST-API) werden immer nur die ersten hundert Treffer angezeigt, da sich in der Datenbank 1.000.000 Einträge befinden und es nicht besonders sinnvoll wäre, (hundert-)tausende von Datensätzen in einem Request zu übertragen bzw. auf der Website darzustellen.

2.7.1 Gestarteter SOAP-Client

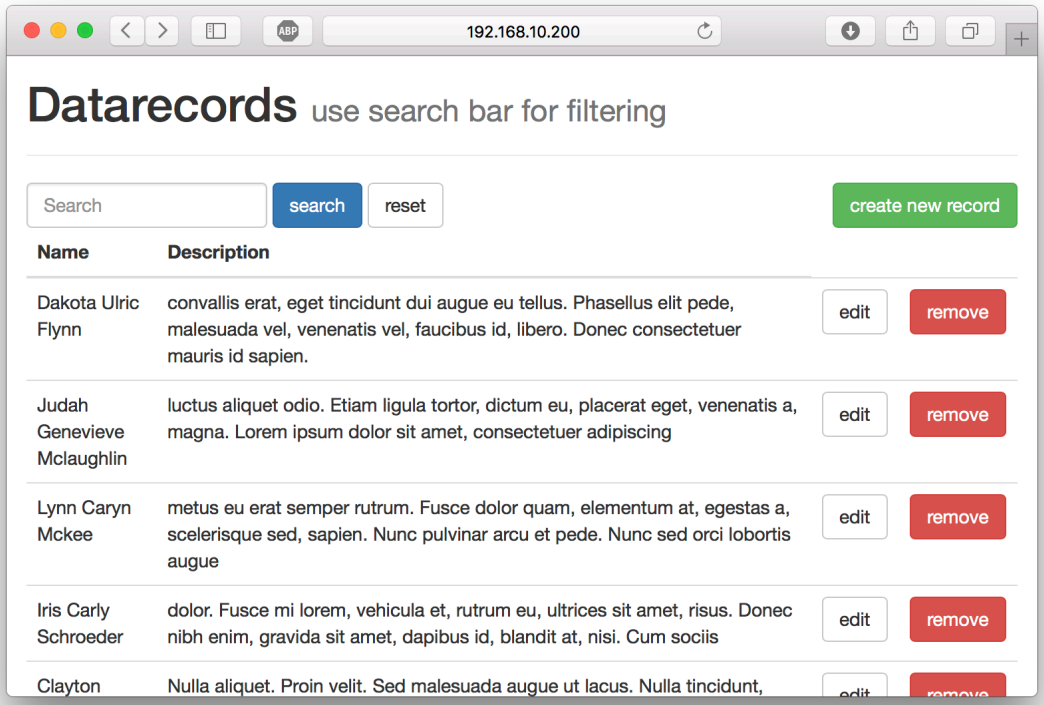
Die Ausgabe des SOAP-Clients sieht beispielsweise folgendermaßen aus:

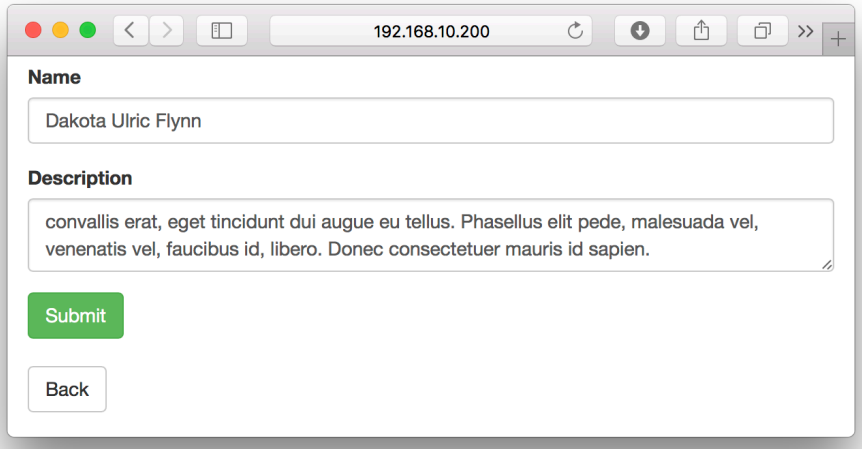
```
<?xml version="1.0" encoding="UTF-16"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:getDataRecordResponse xmlns:ns3="http://at/geyerritter/dezsys07/soa">
            <ns3:dataRecord>
                <id>5672e37578bc38ac63f41a43</id>
                <name>Jenette Ila Guerrero</name>
                <description>metus sit amet ante. Vivamus non lorem vitae odio sagittis semper. Nam tempor diam dictum sapien. Aenean massa. Integer vitae nibh.</description>
            </ns3:dataRecord>
        </ns3:getDataRecordResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die Performance der Suche ist jedenfalls auch mit 1.000.000 Einträge noch gegeben, innerhalb von maximal 2 Sekunden erhält man ein Resultat.

2.7.2 Gestartete Web-Anwendung

Die Web-Anwendung wurde bereits nach Schritt 3 auf `192.168.10.200` hochgeladen. Mittels Browser kann man nun unter dieser URL die Webanwendung aufrufen:





3 Literaturverzeichnis

[1] Generatedata Daten-Generator, Version 3.2.4 [Online]
Verfügbar unter: http://www.generatedata.com
[zuletzt abgerufen am 17.12.2015]

[2] Spring SOAP Web-Service Tutorial (2015) [Online]
Verfügbar unter https://spring.io/guides/gs/producing-web-service/
[zuletzt abgerufen am 17.12.2015]

[3] Vagrant, Development environments made easy [Online]
Verfügbar unter: https://www.vagrantup.com
[zuletzt abgerufen am 17.12.2015]

[4] VirtualBox, Powerful x86 and AMD64/Intel64 virtualization [Online]
Verfügbar unter: https://www.virtualbox.org
[zuletzt abgerufen am 17.12.2015]

[5] Apache Maven Project, software project management and comprehension tool [Online]
Verfügbar unter: https://maven.apache.org
[zuletzt abgerufen am 17.12.2015]

[6] Spring RESTful Web Service Tutorial (2015) [Online]
Verfügbar unter: https://spring.io/guides/gs/rest-service/
[zuletzt abgerufen am 17.12.2015]