

Laborprotokoll

Übungsname

Systemtechnik Labor

5BHIT 2015/16, Gruppe X

Schülername

Version 0.1

Betreuer: Prof. Micheler Begonnen am 2. Oktober 2015

Note: Beendet am 2. Oktober 2015

Inhaltsverzeichnis

[Einführung 3](#_Toc431895375)

[Ziele 3](#_Toc431895376)

[Voraussetzungen 3](#_Toc431895377)

[Aufgabenstellung 3](#_Toc431895378)

[Ergebnisse 4](#_Toc431895379)

# Einführung

Dieses Protokoll-Template soll helfen den Laborübungsteil entsprechend dokumentieren zu können.

## Ziele

Hier werden die zu erwerbenden Kompetenzen und deren Deskriptoren beschrieben. Diese werden von den unterweisenden Lehrkräften vorgestellt.

Dies kann natürlich auch durch eine Aufzählung erfolgen.

## Voraussetzungen

Welche Informationen sind notwendig um die Laborübung reibungslos durchführen zu können? Hier werden alle Requirements der Lehrkraft detailliert beschrieben und mit Quellen untermauert.

Hier zum Beispiel die Architektur der Common Object-Request-Broker Architecture:

## Aufgabenstellung

Hier wird dann die konkrete Aufgabenstellung der Laborübung definiert.

Nun kommt ein Seitenumbruch, um eine klare Trennung der Schülerarbeit zu bestimmen.

# Ergebnisse

Hier sollen die Schritte der Laborübung erläutert werden. Alle Fragestellungen der Lehrkraft müssen hier beantwortet werden. Etwaige Probleme bzw. Schwierigkeiten sollten ebenfalls hier angeführt werden.

Es kann gut möglich sein, dass Lehrkräfte hier auch noch andere Eckpunkte explizit verlangen. Diese können dann in derselben Hierarchiestufe wie die *Ergebnisse* eingeordnet werden.

## 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.

## 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.

## REST-Schnittstelle

### Datenzugriff

Es wurde eine Model Klasse mit drei Feldern (ID, Name und Beschreibung) definiert. Um diese 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 Methode „findTop100ByNameContainingIgnoreCase“ erstellt. 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.

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.

# Literaturverzeichnis

[2] Spring SOAP Web-Service Tutorial (2015) [online]

verfügbar unter <https://spring.io/guides/gs/producing-web-service/>

[zuletzt abgerufen am 17.12.2015]