TGM - HTBLuVA Wien XX
IT Department

Service Oriented Architecture and Restful Webservice

Dezsys 08

Authors: Siegel Hannah & Nachname2 Vorname2

April 12, 2015

Contents

1 Working time

Task	**Person**	**Estimated**	**Final**
Preparation and Frameworks	Haidn	60 minutes	60 minutes
Creating the Knowledge base with Hibernate	Haidn	60 minutes	180 minutes
Creating the Knowledge base	Haidn		60 minutes
Inserting Test-Data	Haidn	30 minutes	30 minutes
Testing the Performance	Haidn	30 minutes	60 minutes
RestFul CRUD Operations	Siegel Haidn	90 minutes 90 minutes	210 minutes 90 minutes
Generate SOA Webservice	Siegel	60 minutes	120 minutes
Include SOAP for SOA Webservice	Siegel	30 minutes	60 minutes
Generate WSDL File	Siegel	30 minutes	10 minutes
Generate Website	Haidn	60 minutes	90 minutes
Generate Client for SOA Webservice	Siegel	60 minutes	180 minutes
Document Datatransfer with SOAP	Siegel	60 minutes	60 minutes
Testing	Siegel Haidn	90 minutes 90 minutes	90 minutes 90 minutes
Documentation	Siegel	60 minutes	60 minutes
Testing	Siegel Haidn	8 hours 7 hours	xxx xxx
Total Team		**15 hours**	**xx hours**

2 Aufgabenstellung

3 Knowledge Base

For the Knowledge Base we decided to use Hibernate and JPA.
We were using IntelliJ and therefore it was quite easy.
At first, we were making a new project, which has already a Hello World Webservice. We copied the hibernate config file from the *Westbahn* project into the `src` folder.
Also, we had to add all the needed libraries, such as the mysql connector and the hibernate libraries.

The Knowledge Base class can be seen in Listing 1.

Listing 1: KnowledgeBase entity class

```
// import statements

@Entity
```

```
public class KnowledgeBase {
    @Id
    @GeneratedValue
    private Long ID;

    @Column(unique=true)
    @Size(min=10, max=500)
    private String text;

    @Column(unique=true)
    @Size(min=10, max=50)
    private String topic;

        // constructor, getter and setter
}
```

- database connection is not working tho -¿ mocking it - new module : JPA module... (file - new module - JPA, Hibernate, SQL Supp)

brauch nun schon 2 stunden fuer hibernate skipping this

4 Build a RESTful WebService

REST is an architectural style which is based on web-standards and the HTTP protocol. REST was first described by Roy Fielding in 2000.In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. In a REST based architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources. Every resource should support the HTTP common operations. Resources are identified by global IDs (which are typically URIs). REST allows that resources have different representations, e.g., text, XML, JSON etc. The REST client can ask for a specific representation via the HTTP protocol (content negotiation) [2]

HTTP methods
The PUT, GET, POST and DELETE methods are typical used in REST based architectures. [2]

4.1 JAX-RS

First we decided to use the JAX-RS libary. It works with annotations which seems really easy. The code could also be generated with IntelliJ but we were not able to find any possibility to deploy it using the IDE and we encountered too many errors. We were also trying an complete tutorial ([2]).Still we were not able to solve any of them, we decided to use another language - PHP. An example for an Hello World JAX-RS Webservice can be found in Listing 2.

Listing 2: Restful Webservice

```
@Path("/helloworld")
```

```
public class RestfulWebservice {
    @GET
    @Produces("text/plain")
    public String getClichedMessage() {
        return "Hello World!";
    }
}
```

5 Building the SOA Webservice in Java

Because we were using the IntelliJ IDEA, this was quite easy. A hello world example with jax-ws can be generated under *new project* ->*Check Webservice* ->Check *Generate server sample code*.
This client code is a Hello World example and can already be run. When we accessed it trough the webbrowser we were able to see some information about the service and the generated wsdl File.

Under [3] we were able to find a complete example, using an interface and more important also a client. So we were changing our code then. We used the hello world example and the tutorial from mkyong to adapt the following classes:

- `Searchable`-Interface (Listing 3)
- `KnowledgeBaseSearcher`-Class which implements the `Searchable`-Interface and provides the search method (Listing 4)
- `KnowledgeBaseSearcherPublisher`-Class which publishes the `KnowledgeBaseSearcher`-Service (Listing 5)
- `KnowledgeBaseSearcherClient`-Class which is using the service over it's WSDL File (Listing 6)

Man muss nun zuerst den `KnowledgeBaseSearcherPublisher` starten, um das Service zu publishen und das Starten des Clients (`KnowledgeBaseSearcherClient`) und das unten angegebene Beispiel nun gibt nun *Return the search query with the search of: blabla* aus.

Listing 3: Searchable interface

```
// imports javax.ws.*
@WebService
@SOAPBinding(style = Style.RPC)
public interface Searchable {
    @WebMethod String search(String searchstring);
}
```

Listing 4: KnowledgeBaseSearcher class

```java
// imports javax.ws.*
@WebService(endpointInterface = "soa.Searchable")
public class KnowledgeBaseSearcher implements Searchable{
    @Override
    public String search(String searchstring) {
        return "Return the search query with the search of: "+searchstring;
    }
```

Listing 5: KnowledgeBaseSearcherPublisher class

```java
public class KnowledgeBaseSearcherPublisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:9999/soa/searcher", new
            KnowledgeBaseSearcher());
    }
}
```

Listing 6: KnowledgeBaseSearcherClient class

```java
public class KnowledgeBaseSearcherClient {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:9999/soa/searcher?wsdl");

        //1st argument service URI, refer to wsdl document above
        //2nd argument is service name, refer to wsdl document above
        QName qname = new QName("http://soa/", "KnowledgeBaseSearcherService");

        Service service = Service.create(url, qname);
        Searchable searcher = service.getPort(Searchable.class);

        System.out.println(searcher.search("blabla"));
    }
}
```

References

[1] **The Java EE 6 Tutorial**
http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html
last used: 06.04.2014, 11:52

[2] **REST with Java (JAX-RS) using Jersey - Tutorial**, Lars Vogel
http://www.vogella.com/tutorials/REST/article.html
last used: 11.04.2014, 13:58

[3] **JAX-WS Hello World Example – RPC Style**, mkyong ,August 29, 2012
http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example/
last used: 12.04.2014, 10:36