Knowledgebase

SOA & REST auf Jetty embedded

Inhaltsverzeichnis

[Projektstruktur 2](#_Toc417251908)

[Embedded Jetty Container 2](#_Toc417251909)

[Interface und REST 2](#_Toc417251910)

[SOA 2](#_Toc417251911)

[Datenbankmodell 3](#_Toc417251912)

[Persistierung 3](#_Toc417251913)

[REST Service – CRUD umsetzung 4](#_Toc417251914)

[Intellij integration 4](#_Toc417251915)

[SOA Service – Umsetzung via jws 5](#_Toc417251916)

[Erstellen eines Interfaces 5](#_Toc417251917)

[Implementierung des Interfaces 5](#_Toc417251918)

[Generierung der wsdl Files 5](#_Toc417251919)

[Intellij integration 5](#_Toc417251920)

[Gradle Build File 6](#_Toc417251921)

[REST Client 7](#_Toc417251922)

[SOA Client 7](#_Toc417251923)

# Projektstruktur

## Embedded Jetty Container

Wir haben uns für Jetty embedded entschieden um keine abhängigkeit auf einen Web Container zu erzeugen sowie um die damit verbundene Konfiguration des Containers in der Applikation zu übernehmen

Der Jetty Container übernimmt das Bereitstellen der REST API, diese ist auf server/rest/\* zu erreichen.

Außerdem stellt Jetty das Interface Servlet auf dem root Verzeichnis zur verfügung.

## Interface und REST

Das Web-Interface sowie REST wird via Servlet eingebunden

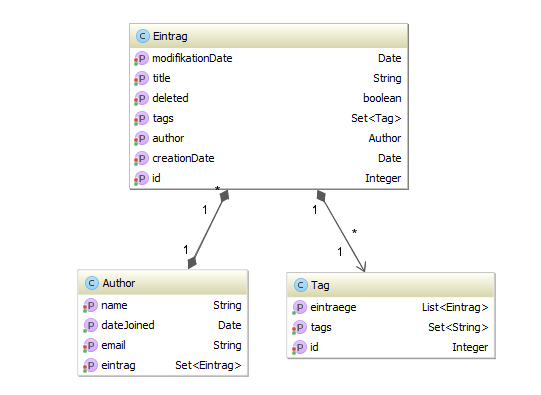
Um die Rest API als Servlet zu verwenden wird Jersey verwendet.

ServletContextHandler handler = **new** ServletContextHandler(server,  
 **"/"**, **true**, **false**);  
*//INTEFACE*ServletHolder servletHolder = **new** ServletHolder(  
 Servlet.**class**);  
handler.addServlet(servletHolder, **"/\*"**);  
server.addConnector(c);  
servletHolder.setInitOrder(0);  
  
*//REST*ServletHolder jerseyServlet = handler.addServlet(  
 org.glassfish.jersey.servlet.ServletContainer.**class**, **"/rest/\*"**);  
jerseyServlet.setInitOrder(0);  
  
*// Tells the Jersey Servlet which REST service/class to load.*jerseyServlet.setInitParameter(  
 **"jersey.config.server.provider.classnames"**,  
 Rest.**class**.getCanonicalName());  
  
server.start();

## SOA

SOA wird via javax.xml.ws.Endpoint auf einem anderen Port zur Verfügung gestellt.

## Datenbankmodell



# Persistierung

Als Datenbank verwenden wir die Relationale Datenbank H2. Diese ist in Java eingebunden und benötigt keinerlei Konfiguration.

Die Daten wurden anfangs via Hibernate eingebunden, es stellte sich heraus das wir auf den EntityManager von Hibernate keinen Zugriff haben, da Jetty-Embedded keine EJBs unterstützt.

# REST Service – CRUD umsetzung

Definition der REST Bean:

@Stateless  
@LocalBean  
@Path(**"/URLThatWillBeAccessed"**)  
**public class** Rest {

Insofern der Container in der Lage ist den java persistence entityManager zu injecten, ist die wie folgt möglich:

@PersistenceContext(unitName=**"CustomerService"**,  
 type=PersistenceContextType.***TRANSACTION***)  
 EntityManager **entityManager**;

Erstellen eines eintrags

@POST  
@Consumes(MediaType.***APPLICATION\_XML***)  
**public void** create(Eintrag eintrag) {

Lesen eines Eintrags via parameter

@GET  
@Produces(MediaType.***APPLICATION\_XML***)  
@Path(**"{name}"**)  
**public** Eintrag read(@PathParam(**"name"**) String name) {

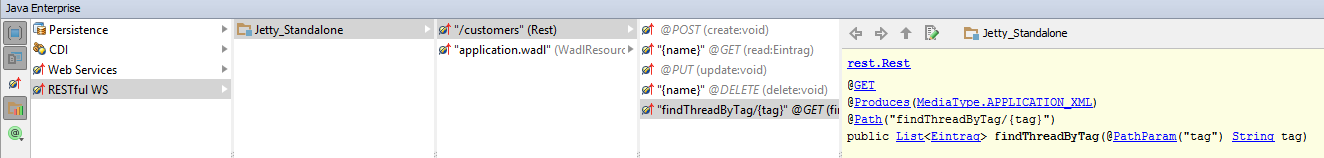
Updaten eines eintrags

@PUT  
@Consumes(MediaType.***APPLICATION\_XML***)  
**public void** update(Eintrag customer) {

Löschen eines Eintrags:

@DELETE  
@Path(**"{name}"**)  
**public void** delete(@PathParam(**"name"**) String name) {

## Intellij integration

Die REST Services sind nun unter dem Reiter Java Enterprise wie folgt zu finden:

# SOA Service – Umsetzung via jws

## Erstellen eines Interfaces

@WebService  
@SOAPBinding(style = Style.***RPC***)  
**public interface** Searchable {  
 @WebMethod  
 String search(String searchFor);  
}

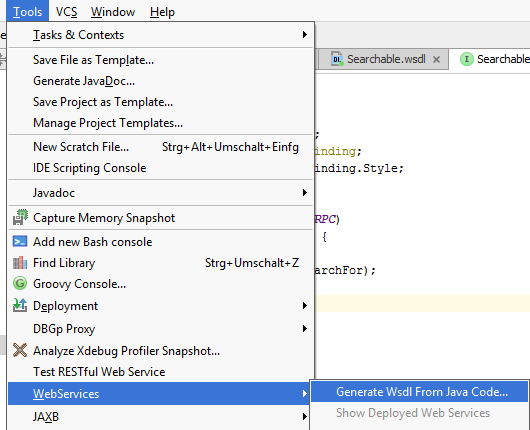
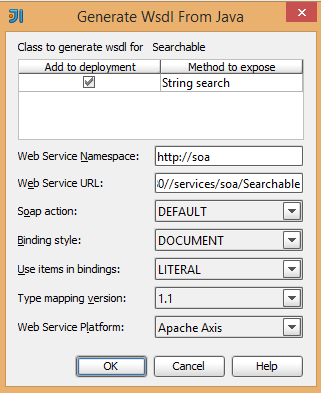
## Implementierung des Interfaces

@WebService(endpointInterface = **"soa.Searchable"**)  
**public class** SOASearcher **implements** Searchable {

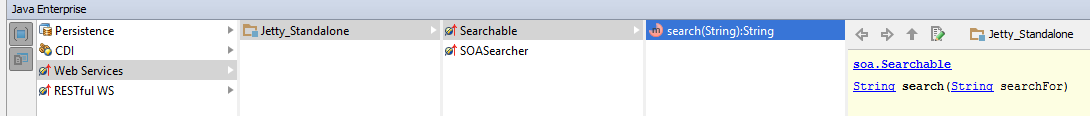
@Override  
**public** String search(String searchFor) {...

## Generierung der wsdl Files

Es wird zur Generierung das Axis Framework von Apache benötigt



## Intellij integration

Die SOA Webservices sind nun unter dem Reiter Java Enterprise wie folgt zu finden:

# Gradle Build File

Um ein Systemunabhängiges bauen des Projekts zu gewährleisten wird gradle benutzt.

**build.gradle**

apply plugin: **'java'**apply plugin: **'application'**apply plugin: **'eclipse'**apply plugin: **'idea'**sourceCompatibility = 1.7  
targetCompatibility = 1.7  
version = **'1.0'**mainClassName = **'Main'**repositories {  
 mavenCentral()  
}  
  
dependencies {  
 *//Standalone Jetty server* compile **'org.eclipse.jetty:jetty-server:9.2.0.RC0'** compile **'org.eclipse.jetty:jetty-servlet:9.2.0.RC0'** *//JEE api* compile **'javax:javaee-api:7.0'** *//Hibernate* compile **'org.hibernate:hibernate-core:4.3.9.Final'** *//H2* compile **'com.h2database:h2:1.4.187'** *//To compile the WSDL* compile **'org.apache.axis:axis:1.4'** compile **'commons-logging:commons-logging:1.2'** compile **'commons-discovery:commons-discovery:0.5'** compile **'org.apache.axis2:axis2-kernel:1.6.2'** *//REST container* compile **'org.glassfish.jersey.media:jersey-media-moxy:2.17'** compile **'org.glassfish.jersey.containers:jersey-container-servlet-core:2.17'** compile **'org.glassfish.jersey.containers:jersey-container-jetty-http:2.17'** compile **'org.glassfish.jersey.core:jersey-server:2.17'**}

# REST Client

# SOA Client