**Step by step how to develop a REST SSL based web service project with Spring and Apache cxf**.

Prerequisite for this project are:-

1.       Knowledge about spring framework <https://en.wikipedia.org/wiki/Spring_Framework>

2.       Knowledge about JAX-RS <https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services>

3.       Json message structure <https://en.wikipedia.org/wiki/JSON>

4.       Knowledge of Eclipse.

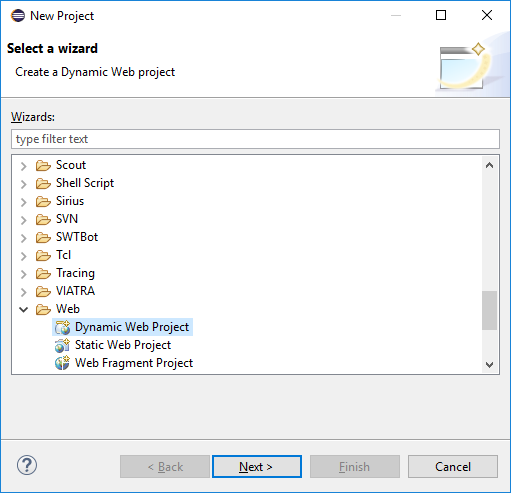
5.       Also down load IBM Liberty for developer.

Here I have referring to Wikipedia documentation only. We can read any documentation on all the above topic which you think more valuable for you.

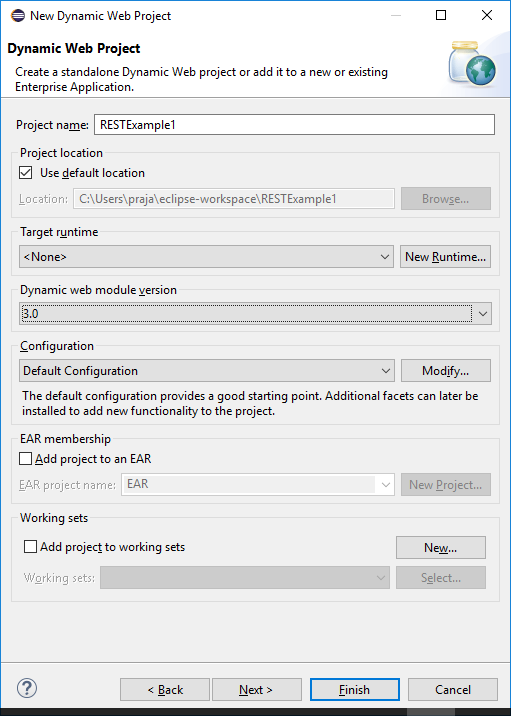
The aim of this small project is to help developer/student to develop a REST web service application with Spring and Jersey framework. You are open to add any advice or example here. Based on your advice I will keep updating it in future.

We are using json messaging in this example.

1. First create a Dynamic web project

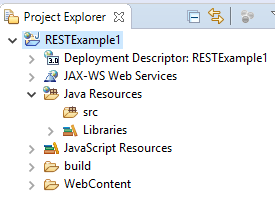


Click next and name it RESTExample1

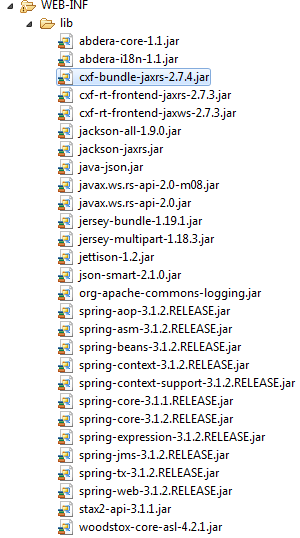


And click ‘Finish’

Now you can see a project created like below:-

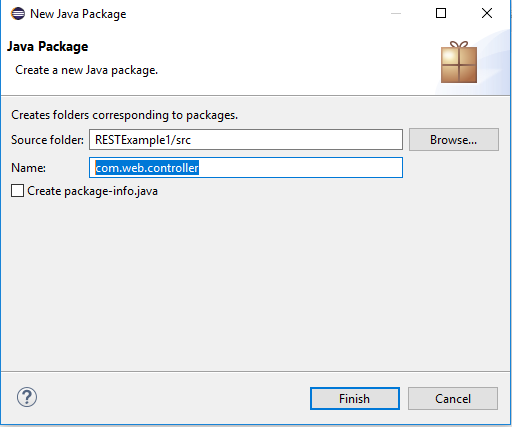


1. Expand ‘WebContent’ folder and add below jars in the project lib folder ( you can down load these jar files online. Just type download followed by jar name for example download abdera-core-1.1.jar in google search engine. Better down load from mavan site. (It is not mandatory to add all these jar files in lib folder. You can deploy these jars in your server and add all of them in your class path)

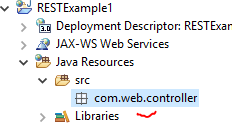


1. Now create package explained below and add interface into the project

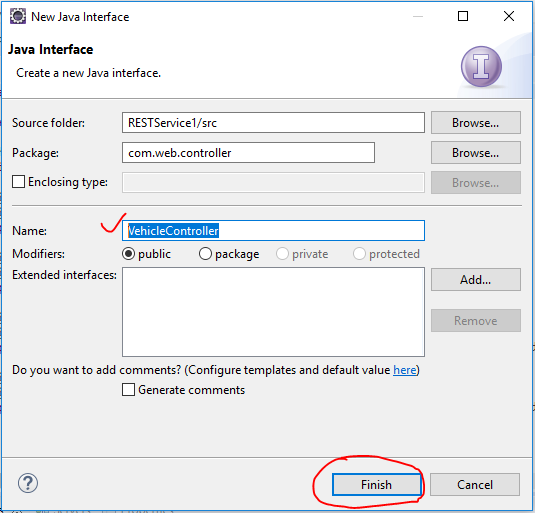
Right click on project RESTExample1 from window highlight new and from the sub-window select package and enter below name and click ‘Finish’ button



Now you can see a empty package like below



Now right click on the package ‘com.web.controller’ and select ‘New’ and from the sub-window select interface and provide the name as VehicleController



Add the below code into the empty interface. (in the interface file press ‘ctrl+A’ and copy and paste below code – just replace all auto generated code with the below code, **but make sure you will comment out both the POST method and its implementation in the class till you complete GET method call test**)

**package** com.web.controller;

**import** javax.ws.rs.Consumes;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.HeaderParam;

**import** javax.ws.rs.POST;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.Context;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.UriInfo;

**import** com.rest.car.bean.CarDetailBean;

**import** com.rest.car.bean.CarInfoBean;

@Path("/v/vehicle")

**public** **interface** VehicleController {

//Below four different get examples

@GET

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response getVehicles();

@GET @Path("/{id}")

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response getVehicleById(@PathParam("id") **int** id);

@GET @Path("/{id}/{id2}")

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response getVehicleById(@Context UriInfo info, @PathParam("id") **int** id, @PathParam("id2") **int** id2);

@GET @Path("/{id}/car/{zone}")

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response getVehicleById(@Context UriInfo info, @PathParam("id") **int** id, @PathParam("zone") String zone,@HeaderParam("user") String user);

//Below two different POST examples

//comment out below two methods for time being

@POST

@Produces(MediaType.***APPLICATION\_JSON***)

@Consumes(MediaType.***APPLICATION\_JSON***)

@Path("/orders")

**public** Response registerCar(@Context UriInfo info, CarDetailBean carDetailBean, @HeaderParam("user") String user);

@POST

@Produces(MediaType.***APPLICATION\_JSON***)

@Consumes(MediaType.***APPLICATION\_JSON***)

@Path("/orderlist/vehicle.json")

**public** Response placeCarOrder(@Context UriInfo info, CarInfoBean carInfoBean, @HeaderParam("user") String user);

}

Please look at the different @Path declaration above. The intention behind declaring different path here is to explain you different examples of @Path declaration by which you can design your own REST webservices as per your requirement.

1. Create below controller class and implements the above interface (follow above steps and select class this time instead of interface, to know more about controller class check online materials)

----------------------------------------------------------------------

//copy from here to

**package** com.web.controller.impl;

**import** java.io.StringWriter;

**import** java.io.Writer;

**import** java.util.List;

**import** net.minidev.json.JSONObject;

**import** org.codehaus.jackson.map.ObjectMapper;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.MultivaluedMap;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.UriInfo;

**import** javax.ws.rs.core.Response.Status;

**import** com.web.data.Vehicle;

**import** com.web.data.VehicleStore;

**import** com.rest.car.bean.CarDetailBean;

**import** com.rest.car.bean.CarInfoBean;

**import** com.web.controller.VehicleController;

**public** **class** VehicleControllerImpl **implements** VehicleController{

ObjectMapper JSON\_MAPPER = **new** ObjectMapper();

String result = **null**;

@Override

**public** Response getVehicles(){

System.***out***.println("In getVehicles()");

VehicleStore vehicleStore = VehicleStore.*getStoreInstance*();

List<Vehicle> listOfVehicles = vehicleStore.getVehicleList();

JSONObject obj=**new** JSONObject();

**try**{

obj.put("Vehicles", listOfVehicles);

}**catch**(Exception je){

je.printStackTrace();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity(je.getMessage()).type(MediaType.***APPLICATION\_JSON***).build();

}

System.***out***.println("Exit getVehicles()");

**return** Response.*status*(Status.***OK***).entity(obj).type(MediaType.***APPLICATION\_JSON***).build();

}

@Override

**public** Response getVehicleById(**int** id){

System.***out***.println("In getVehicleById(int id)");

System.***out***.println("id :"+id);

VehicleStore vehicleStore = VehicleStore.*getStoreInstance*();

List<Vehicle> listOfVehicles = vehicleStore.getVehicleList();

JSONObject obj=**new** JSONObject();

**for** (Vehicle vehicle: listOfVehicles) {

**if**(vehicle.getId()==id){

**try**{

obj.put("Vehicle", vehicle);

**return** Response.*status*(Status.***OK***).entity(obj).type(MediaType.***APPLICATION\_JSON***).build();

}**catch**(Exception je){

je.printStackTrace();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity(je.getMessage()).type(MediaType.***APPLICATION\_JSON***).build();

}

}

}

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity("No Data found").type(MediaType.***APPLICATION\_JSON***).build();

}

@Override

**public** Response getVehicleById(UriInfo info, **int** id1, **int** id2){

System.***out***.println("In getVehicleById(UriInfo info, int id1, int id2)");

System.***out***.println("id1 :"+id1);

System.***out***.println("id2 :"+id2);

MultivaluedMap <String, String> parameters = info.getQueryParameters();

String userID = parameters.getFirst("user");

String srvcURL = info.getRequestUri().toString();

System.***out***.println("srvcURL : "+srvcURL);

VehicleStore vehicleStore = VehicleStore.*getStoreInstance*();

List<Vehicle> listOfVehicles = vehicleStore.getVehicleList();

JSONObject obj=**new** JSONObject();

**boolean** vehicleFound = **false**;

**for** (Vehicle vehicle: listOfVehicles) {

**if**(vehicle.getId()==id1 || vehicle.getId()==id2){

**try**{

obj.put("Vehicle"+vehicle.getId(), vehicle);

vehicleFound = **true**;

}**catch**(Exception e){

e.printStackTrace();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity(e.getMessage()).type(MediaType.***APPLICATION\_JSON***).build();

}

}

}

**if**(vehicleFound)

**return** Response.*status*(Status.***OK***).entity(obj).type(MediaType.***APPLICATION\_JSON***).build();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity("No Data found").type(MediaType.***APPLICATION\_JSON***).build();

}

@Override

**public** Response getVehicleById(UriInfo info, **int** id,String zone,String user){

System.***out***.println("In getVehicleById(UriInfo info, int id,String zone,String user)");

System.***out***.println("id : "+id);

System.***out***.println("user : "+user);

System.***out***.println("Zone : "+zone);

VehicleStore vehicleStore = VehicleStore.*getStoreInstance*();

List<Vehicle> listOfVehicles = vehicleStore.getVehicleList();

JSONObject obj=**new** JSONObject();

**for** (Vehicle vehicle: listOfVehicles) {

**if**(vehicle.getId()==id){

**try**{

vehicle.setCountryName("Vehicle From :" + zone);

obj.put("Vehicle", vehicle);

Writer strWriter = **new** StringWriter();

JSON\_MAPPER.writeValue(strWriter, vehicle);

result = JSON\_MAPPER.writeValueAsString(strWriter);

**return** Response.*status*(Status.***OK***).entity(result).type(MediaType.***APPLICATION\_JSON***).build();

}**catch**(Exception e){

e.printStackTrace();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity(e.getMessage()).type(MediaType.***APPLICATION\_JSON***).build();

}

}

}

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity("No Data found").type(MediaType.***APPLICATION\_JSON***).build();

}

@Override

**public** Response registerCar(UriInfo info, CarDetailBean carDetailBean, String user) {

System.***out***.println("In registerCar(UriInfo info, CarDetailBean carDetailBean, String user)");

VehicleStore vehicleStore = VehicleStore.*getStoreInstance*();

List<Vehicle> listOfVehicles = vehicleStore.getVehicleList();

**for** (Vehicle vehicle: listOfVehicles) {

**if**(vehicle.getId()==carDetailBean.getCarId()){

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity("Car id already present. Please use different id").type(MediaType.***APPLICATION\_JSON***).build();

}

}

System.***out***.println("In registerCar(UriInfo info, CarDetailBean carDetailBean, String user)2");

Vehicle newvehicle = **new** Vehicle(carDetailBean.getCarId(), carDetailBean.getCarInfo());

System.***out***.println("In registerCar(UriInfo info, CarDetailBean carDetailBean, String user)3");

vehicleStore.addVehicle(newvehicle);

System.***out***.println("In registerCar(UriInfo info, CarDetailBean carDetailBean, String user)4");

JSONObject obj=**new** JSONObject();

**try**{

obj.put("Vehicles", listOfVehicles);

}**catch**(Exception je){

je.printStackTrace();

**return** Response.*status*(Status.***EXPECTATION\_FAILED***).entity(je.getMessage()).type(MediaType.***APPLICATION\_JSON***).build();

}

System.***out***.println("Exit getVehicles()");

**return** Response.*status*(Status.***OK***).entity(obj).type(MediaType.***APPLICATION\_JSON***).build();

}

@Override

**public** Response placeCarOrder(UriInfo info, CarInfoBean carInfoBean, String user) {

System.***out***.println("In placeCarOrder(UriInfo info, CarInfoBean carInfoBean, String user)");

JSONObject obj=**new** JSONObject();

obj.put("CarOrder", carInfoBean);

**return** Response.*status*(Status.***OK***).entity(obj).type(MediaType.***APPLICATION\_JSON***).build();

}

}

//copy till this line

1. Now add/create a bean class call Vehicle as below

//copy from here

**package** com.web.data;

**import** java.io.Serializable;

**import** org.codehaus.jackson.annotate.JsonIgnoreProperties;

**import** org.codehaus.jackson.annotate.JsonProperty;

**import** org.codehaus.jackson.annotate.JsonPropertyOrder;

@JsonPropertyOrder({"id", "VehicleName"})

@JsonIgnoreProperties(ignoreUnknown = **true**)

**public** **class** Vehicle **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = -3501772243949297054L;

@JsonProperty("id")

**private** **int** id;

@JsonProperty("vehicleName")

**private** String vehicleName;

@JsonProperty("countryName")

**private** String countryName;

**public** Vehicle(){};

**public** Vehicle(**int** i, String vehicleName){

**super**();

**this**.id = i;

**this**.vehicleName = vehicleName;

}

**public** **int** getId(){

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getVehicleName() {

**return** vehicleName;

}

**public** **void** setVehicleName(String vehicleName) {

**this**.vehicleName = vehicleName;

}

**public** String getCountryName() {

**return** countryName;

}

**public** **void** setCountryName(String countryName) {

**this**.countryName = countryName;

}

}

//copy till here

1. Now create a singletone class as mentioned below

//copy from here

**package** com.web.data;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** VehicleStore {

**static** VehicleStore *vehicleStore* = **null**;

List<Vehicle> \_listOfVehicles = **null**;

**private** VehicleStore() {

\_listOfVehicles = createVehicleList();

}

**public** **static** VehicleStore getStoreInstance() {

**if**(*vehicleStore* == **null**) {

*vehicleStore* = **new** VehicleStore();

}

**return** *vehicleStore*;

}

**public** **void** addVehicle(Vehicle vehicle) {

\_listOfVehicles.add(vehicle);

}

**public** List<Vehicle> getVehicleList(){

**return** \_listOfVehicles;

}

**public** List<Vehicle> createVehicleList(){

System.***out***.println("Creating Vehicle List");

Vehicle suzukiVehicle=**new** Vehicle(1, "Swif");

Vehicle bmwVehicle=**new** Vehicle(4, "BMW");

Vehicle volvoVehicle=**new** Vehicle(3, "Volvo");

Vehicle jeepVehicle=**new** Vehicle(2, "Jeep");

List<Vehicle> listOfVehicles = **new** ArrayList<Vehicle>();

listOfVehicles.add(suzukiVehicle);

listOfVehicles.add(bmwVehicle);

listOfVehicles.add(volvoVehicle);

listOfVehicles.add(jeepVehicle);

**return** listOfVehicles;

}

}

//copy till here

1. Now create two handler class as below:-

(First create package , ‘com.web.cxf.in.handler’ and add class CxfInPutHandler)

[Handler class plays a very inportant role. They are intercepters. First one is for all incoming request and second one is for all outgoing response. During test just check the logs or put break point to check how it work. A inhandler will receive all webservice call first before it divert it to actual web method to call. Therefore you can use this class for any authentication purpose or any other activities which you think execute first before the control hit actual webservice method. Please see online materials for detail.]

**//copy from here**

**package** com.web.cxf.in.handler;

**import** javax.servlet.http.HttpServletRequest;

**import** org.apache.cxf.interceptor.Fault;

**import** org.apache.cxf.message.Message;

**import** org.apache.cxf.phase.AbstractPhaseInterceptor;

**import** org.apache.cxf.phase.Phase;

**import** org.apache.cxf.transport.http.AbstractHTTPDestination;

**public** **class** CxfInPutHandler **extends** AbstractPhaseInterceptor<Message> {

**public** CxfInPutHandler(){

**super**(Phase.*PRE\_STREAM*);

}

@Override

**public** **void** handleMessage(**final** Message message) **throws** Fault {

**try** {

System.*out*.println("In in handleMessage() ");

String user = "";//header param

String pid = "";//header param

String pathInfo = "";

**if** (message.getExchange().getInMessage() != **null**) {

**if** (message.getExchange().getInMessage().get(AbstractHTTPDestination.*HTTP\_REQUEST*) != **null**) {

HttpServletRequest req = (HttpServletRequest) message.getExchange().getInMessage().get(AbstractHTTPDestination.*HTTP\_REQUEST*);

user = req.getHeader("user");

pid = req.getHeader("pid");

pathInfo = req.getPathInfo();

}

} **else** {

System.*out*.println("Fail to get the http request ");

}

}**catch**(Exception e){

e.printStackTrace();

}

}

}

And below out-handler. All the webservice response (before returning to caller) will hit this service. Just think what business logic you would like to implement here.

**package** com.web.cxf.out.handler;

**import** javax.servlet.http.HttpServletRequest;

**import** org.apache.cxf.interceptor.Fault;

**import** org.apache.cxf.message.Message;

**import** org.apache.cxf.phase.AbstractPhaseInterceptor;

**import** org.apache.cxf.phase.Phase;

**import** org.apache.cxf.transport.http.AbstractHTTPDestination;

**public** **class** CxfOutPutHandler **extends** AbstractPhaseInterceptor<Message> {

**public** CxfOutPutHandler() {

**super**(Phase.*PRE\_STREAM*);

}

@Override

**public** **void** handleMessage(**final** Message message) **throws** Fault {

System.*out*.println("In out handleMessage() ");

**if** (message.getExchange().getInMessage() != **null**) {

**if** (message.getExchange().getInMessage().get(AbstractHTTPDestination.*HTTP\_REQUEST*) != **null**) {

HttpServletRequest req = (HttpServletRequest) message.getExchange().getInMessage().get(AbstractHTTPDestination.*HTTP\_REQUEST*);

//origin = req.getHeader("Origin");

}

} **else** {

System.*out*.println("Fail to get the http request ");

}

}

}

//copy till here

During testing just check the log information to understand how control flows through the code. See the below log information:-

In in handleMessage() (hit in handler interceptor before calling getVehicles)

In getVehicles()

Creating Vehicle List

Exit getVehicles()

In Out handleMessage() (finally hit out interceptor )

Now we are going to introduce one Filter here. Control will hit the Filter first even before input handler. It works same as handler. Read more about Filter in web project:-

Just following the previous instruction and add below Filter class:-

package com.web.filter;

/\*

\* A filter is typically used to perform a particular piece of functionality

either before or after the primary functionality of a web application is

performed. As an example, if a request is made for a particular resource such

as a servlet and a filter is used, the filter code may execute and then pass

the user on to the servlet. As a further example, the filter might determine

that the user does not have permissions to access a particular servlet, and it might

send the user to an error page rather than to the requested resource.

\*/

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletRequestWrapper;

public class UserFilter implements Filter{

@Override

public void destroy() {

}

/\*

\* Here I am just setting a header field before control goes to the servlet.

\* We can also authenticate a user here before re-directing to a servlet

\*/

@Override

public void doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) throws IOException, ServletException {

System.out.println("Im in UserFilter");

HttpServletRequest req = (HttpServletRequest) request;

HeaderRequestWrapper requestWrapper = new HeaderRequestWrapper(req);

requestWrapper.addHeader("user", "Test1234");

chain.doFilter(requestWrapper, response);

}

@Override

public void init(FilterConfig filterConfig) throws ServletException {

}

public class HeaderRequestWrapper extends HttpServletRequestWrapper {

public HeaderRequestWrapper(HttpServletRequest request) {

super(request);

}

private Map<String, String> headerMap = new HashMap<String, String>();

public void addHeader(String name, String value) {

headerMap.put(name, value);

}

@Override

public String getHeader(String name) {

String headerValue = super.getHeader(name);

if (headerMap.containsKey(name)) {

headerValue = headerMap.get(name);

}

return headerValue;

}

}

}

For filter you have to add below definition in web.xml file

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>com.web.filter.UserFilter</filter-class>

</filter>

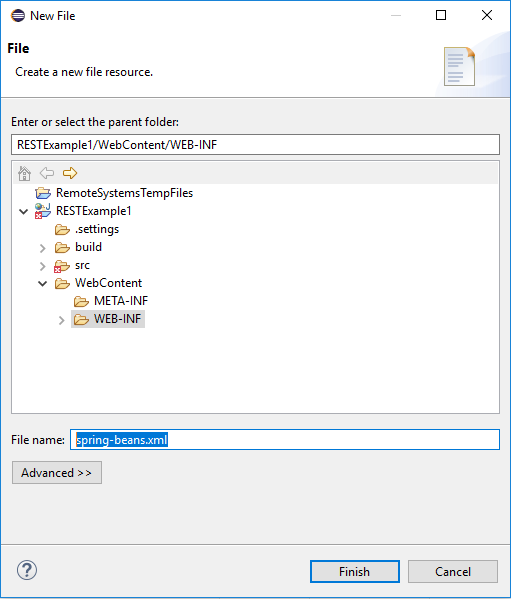
<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

1. Now create a spring beans definition file ‘spring-beans.xml’ as below under WEB-INF



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:cxf=*"http://cxf.apache.org/core"*

xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:cache=*"http://www.springframework.org/schema/cache"* xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd*

*http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd*

*http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd"*>

<import resource=*"classpath:META-INF/cxf/cxf.xml"* />

<import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"* />

<jaxrs:server id=*"vehicleservice"* address=*"/rest/example"*>

<jaxrs:serviceBeans>

<ref bean=*"vehicleController"* />

</jaxrs:serviceBeans>

<jaxrs:providers>

<ref bean=*"jsonProvider"* />

</jaxrs:providers>

</jaxrs:server>

<bean id=*"vehicleController"* class=*"com.web.controller.impl.VehicleControllerImpl"*>

<description>

This Bean contains REST methods.

</description>

</bean>

<bean id=*"jsonProvider"* class=*"org.codehaus.jackson.jaxrs.JacksonJsonProvider"*>

<description>

This Bean contains implementation to serilized json object.

</description>

</bean>

<bean id=*"cxfOutPutHandler"* class=*"com.web.cxf.out.handler.CxfOutPutHandler"*>

<description>

This Bean implements Interceptor for all out going responses

</description>

</bean>

<bean id=*"cxfInputHandler"* class=*"com.web.cxf.in.handler.CxfInPutHandler"* >

<description>

This Bean implements Interceptor for all in coming request

</description>

</bean>

<cxf:bus>

<cxf:outInterceptors>

<ref bean=*"cxfOutPutHandler"* />

</cxf:outInterceptors>

<cxf:inInterceptors>

<ref bean=*"cxfInputHandler"* />

</cxf:inInterceptors>

</cxf:bus>

</beans>

1. Now put the reference in the web deployment file web.xml

(right click on the project and go to Java EE Tools and from the sub-window select ‘Generate deployment descriptor stub’. It will generate web.xml file and in that file after line ‘</welcome-file-list>’add the below lines and save the file.

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>CXFServlet</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

<init-param>

<param-name>disable-address-updates</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>CXFServlet</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/spring-beans.xml

</param-value>

</context-param>

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>com.web.filter.UserFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

Now you are ready to test all the GET web service calls

You can use two approaches here. One is to generate a war or ear package and deploy it in your Web Server and second approach is, just right click on your project and select ‘Run As’ and follows by ‘Run on Server’. Make sure that you have added server to your eclipse already. If not, check online materials how to add a server to eclipse. You can add either Apache Tomcat or IBM Liberty for your testing purpose.

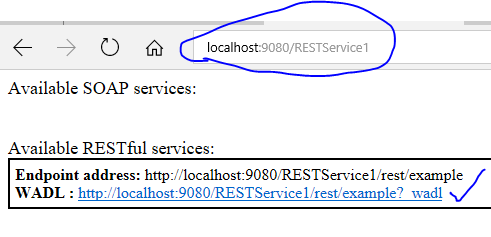
After successful deployment:-

Just copy and paste below url in the address bar of any browser and see the response

<http://host:port/RESTService1>

for me it’s http://localhost:9080/RESTService1

now you can see below in your browser



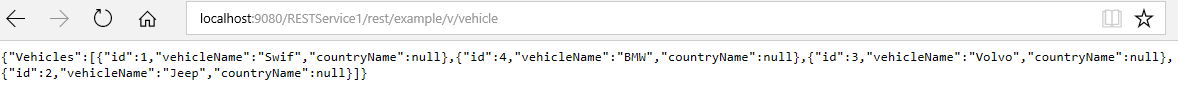
Now just click on the link above you can see all the web services available in it

<?xml version="1.0"?>

[<application xmlns:xs="**http://www.w3.org/2001/XMLSchema**" xmlns="**http://wadl.dev.java.net/2009/02**">](http://localhost:9080/RESTService1/rest/example?_wadl)<grammars/>[<resources base="**http://localhost:9080/RESTService1/rest/example**"><resource path="**/v/vehicle**"><method name="**GET**"><response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method>[<resource path="**/orderlist/vehicle.json**"><method name="**POST**"><request>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/><param name="**user**" type="**xs:string**" style="**header**"/></request>[<response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method></resource>[<resource path="**/orders**"><method name="**POST**"><request>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/><param name="**user**" type="**xs:string**" style="**header**"/></request>[<response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method></resource>[<resource path="**/{id}**">](http://localhost:9080/RESTService1/rest/example?_wadl)<param name="**id**" type="**xs:int**" style="**template**"/>[<method name="**GET**">](http://localhost:9080/RESTService1/rest/example?_wadl)<request/>[<response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method></resource>[<resource path="**/{id}/vehicle/{zone}**">](http://localhost:9080/RESTService1/rest/example?_wadl)<param name="**id**" type="**xs:int**" style="**template**"/><param name="**zone**" type="**xs:string**" style="**template**"/>[<method name="**GET**"><request>](http://localhost:9080/RESTService1/rest/example?_wadl)<param name="**user**" type="**xs:string**" style="**header**"/></request>[<response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method></resource>[<resource path="**/{id}/{id2}**">](http://localhost:9080/RESTService1/rest/example?_wadl)<param name="**id**" type="**xs:int**" style="**template**"/><param name="**id2**" type="**xs:int**" style="**template**"/>[<method name="**GET**">](http://localhost:9080/RESTService1/rest/example?_wadl)<request/>[<response>](http://localhost:9080/RESTService1/rest/example?_wadl)<representation mediaType="**application/json**"/></response></method></resource></resource></resources></application>

Now let test all the GET REST calls:- (Comment out both the POST web services to avoid compilation error)

<http://localhost:9080/RESTService1/rest/example/v/vehicle>



In some web browser it will allow you to download a json file instead of displaying the message in the browser itself. Just down load the json file and open it.

Test rest of all by yourself

<http://localhost:9080/RESTService1/rest/example/v/vehicle/1> (2,3 and 4)

<http://localhost:9080/RESTService1/rest/example/v/vehicle/1/3> (use different combination of numbers from 1 to 4)

<http://localhost:9080/RESTService1/rest/example/v/vehicle/1/car/India>

Let try for POST REST service calls

First un-comment first POST method and add below bean into your project including packages

(If you are uncomfortable to test GET and POST separately, first complete whole code for GET and POST calls and compile and test both at the same time)

**package** com.rest.car.bean;

**import** org.codehaus.jackson.annotate.JsonProperty;

**public** **class** CarDetailBean {

@JsonProperty("carId")

**private** **int** carId;

@JsonProperty("carInfo")

**private** String carInfo;

**public** **int** getCarId() {

**return** carId;

}

**public** **void** setCarId(**int** carId) {

**this**.carId = carId;

}

**public** String getCarInfo() {

**return** carInfo;

}

**public** **void** setCarInfo(String carInfo) {

**this**.carInfo = carInfo;

}

}

Now compile the code and deploy it into your web server or right click on the project and select run as and followed by run on server and select the deployed server from the list.

For post web service call you cannot use a browser. Best use SOAPUI tool for the same or create your own client application. To test with SOAP tool please go to the end of this lesion.

Now use below URL and json message body to test the first POST call

<http://localhost:9080/RESTService1/rest/example/v/vehicle/orders>

{

"carId": 5,

"carInfo": "Safary"

}

The above json body will be injected into the bean class declared above by the json framework.

Let implement the second POST call and test it.

First add all the below beans into your project. This one is bit complex. The reason behind using this complex architecture is to help you to understand how you can manage different complex design as per your requirement. Here in each bean I am implementing a List along with an object. In the first bean ‘CarInfoBean’ I am using one object call CustomerRequestInfo. There will be more than one CustomerRequestInfo hence we are using a List to store it.

package com.rest.car.bean;

import java.util.List;

import org.codehaus.jackson.annotate.JsonProperty;

public class CarInfoBean {

@JsonProperty("userId")

private String userId;

@JsonProperty("passWord")

private String passWord;

@JsonProperty("customerRequestInfo")

private List<CustomerRequestInfo> customerRequestInfo;

public String getUserId() {

return userId;

}

public void setUserId(String userId) {

this.userId = userId;

}

public String getPassWord() {

return passWord;

}

public void setPassWord(String passWord) {

this.passWord = passWord;

}

public List<CustomerRequestInfo> getCustomerRequestInfo() {

return customerRequestInfo;

}

public void setCustomerRequestInfo(List<CustomerRequestInfo> customerRequestInfo) {

this.customerRequestInfo = customerRequestInfo;

}

}

Now second bean i.e. CustomerRequestInfo here also I am declaring one more object call OrderDetail. Intension is very clear so that you can even design more complex as per your requirement.

**//copy from here**

**package** com.rest.car.bean;

**import** java.util.List;

**import** org.codehaus.jackson.annotate.JsonProperty;

**public** **class** CustomerRequestInfo {

@JsonProperty("requestId")

**private** String requestId;

@JsonProperty("orderDetail")

**private** List<OrderDetail> orderDetail;

**public** String getRequestId() {

**return** requestId;

}

**public** **void** setRequestId(String requestId) {

**this**.requestId = requestId;

}

**public** List<OrderDetail> getOrderDetail() {

**return** orderDetail;

}

**public** **void** setOrderDetail(List<OrderDetail> orderDetail) {

**this**.orderDetail = orderDetail;

}

}

And finally OrderDetail bean:-

**//copy from here**

**package** com.rest.car.bean;

**import** org.codehaus.jackson.annotate.JsonProperty;

**public** **class** OrderDetail {

@JsonProperty("carID")

**private** String carID;

@JsonProperty("carModule")

**private** String carModule;

@JsonProperty("carColor")

**private** String carColor;

@JsonProperty("customerComment")

**private** String customerComment;

**public** String getCarID() {

**return** carID;

}

**public** **void** setCarID(String carID) {

**this**.carID = carID;

}

**public** String getCarModule() {

**return** carModule;

}

**public** **void** setCarModule(String carModule) {

**this**.carModule = carModule;

}

**public** String getCarColor() {

**return** carColor;

}

**public** **void** setCarColor(String carColor) {

**this**.carColor = carColor;

}

**public** String getCustomerComment() {

**return** customerComment;

}

**public** **void** setCustomerComment(String customerComment) {

**this**.customerComment = customerComment;

}

}

Now use below url and json body to test this POST call. Also look carefully how the framework injects value into its respective field. To understand detail about json please read online materials.

http://localhost:9080/RESTService1/rest/example/v/vehicle/orderlist/vehicle.json

{

"userId" : "97912345",

"passWord":"Xlcb#11",

"customerRequestInfo":[

{

"requestId":"12345",

"orderDetail":[

{

"carID":"1",

"carModule":"IM3",

"carColor":"RED",

"customerComment":"Should be a right hand car"

},

{

"carID":"7",

"carModule":"BM3",

"carColor":"BLUE",

"customerComment":"Should be a Left hand car"

},

{

"carID":"5",

"carModule":"MM3",

"carColor":"PINK",

"customerComment":"Should be a Left hand car"

}

]

},

{

"requestId":"12346",

"orderDetail":[

{

"carID":"2",

"carModule":"XB1",

"carColor":"RED",

"customerComment":"Should be a right hand car"

},

{

"carID":"3",

"carModule":"XM3",

"carColor":"BLUE",

"customerComment":"Should be a Left hand car"

},

{

"carID":"4",

"carModule":"XX2",

"carColor":"PINK",

"customerComment":"Should be a Left hand car"

}

]

}

]

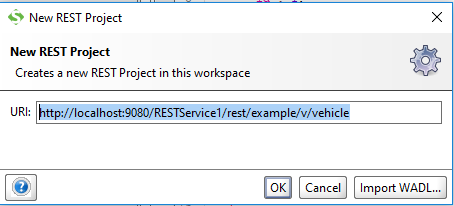
}

My advice is to play around the code so that you can understand the code better. Try to retrieve different values from carInfoBean object and print it.

1. First down load soapUI from the below url and install it into your computer.

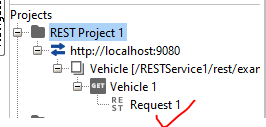
<https://www.soapui.org/downloads/latest-release.html>

1. Open the application and from the file menu select new REST service project and in the window past your url (first example for a GET web service call)

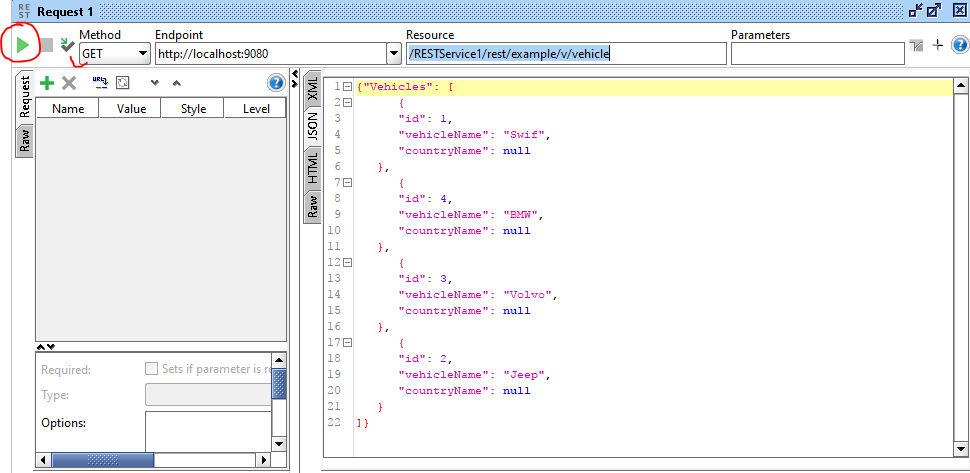


And click ‘OK’

1. Now from project explorer window double click Request1

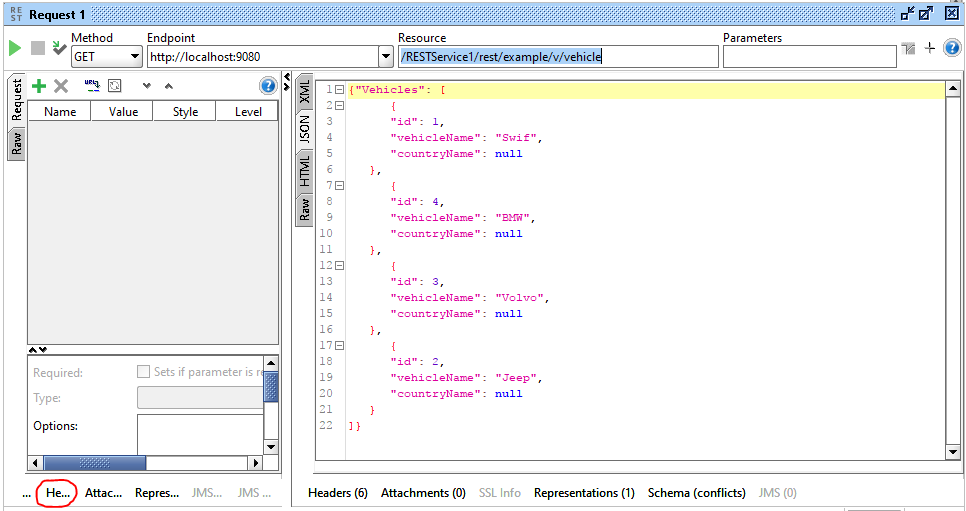


And click the green arrow at the left hand site

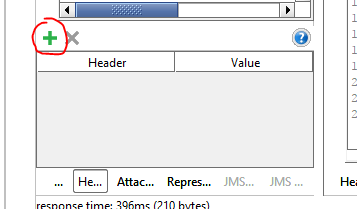


If you need to set some header parameter for your test do it as below:-

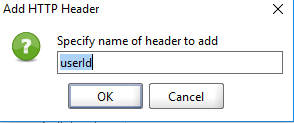
Select Header as below:-



Now click on plus button

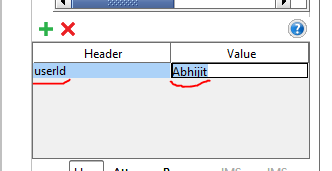


Now add your header parameter as below



And click ‘OK’ button

Now you can see header parameter ‘userId’ has been added to your project now add value for it



SSL based REST service in Java with Spring.

Here prerequisite for you is to read my first article in <https://dzone.com/articles/step-by-step-how-to-develop-a-rest-web-service-pro>. This is a continuation of that article and to understand it better, you should also have the knowledge of below mentioned topics .

1.       What is a ssl certificate? <https://en.wikipedia.org/wiki/Public_key_certificate>

2.       What is SSL handshake? <https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm>

3.       What is java key store (jks)? There are lots of documentations on line, read which is most suitable for you

4.       What is java trust store(jks)

5.       What is PKCS certificate/kestore.

6.       What is HTTPS network protocol? <https://en.wikipedia.org/wiki/HTTPS>

This is going to be a complete SSL based REST project for you. You can download the project from [https://github.com/prateekparallel/JavaSSLRESTWebservice]. You just need to refractor this framework/code base and replace existing java classes with your classes to make this frame work to fulfill your business requirement. You can even use this frame work as your business project as . You can utilize this project as back-end or middle tier application. I am going to add few more new classes to my earlier project to make it a SSL based middle tier (client) application. Also going to add a thread which can be used for health check purpose. In addition to it I am going to introduce one servlet as well.

Already I have explained in my previous project/article how to create package and class in eclipse with screen shots . Therefore here I am directly going to add classes into the existing code base.

1st Java class I am going to add here is the health check thread class. First create package ‘com.web.system.monitor’ in your existing project and add below class ‘MonitorHeartBeat’:-

/\*\*\*\*

\* This is a health check thread example. You can add more threads like this

\* in your application as per your requirement. This class will spawns

\* a thread at the very beginning, when this web service application will start loading.

\* This thread will be instantiated from startup servlet.

\*/

**package** com.web.system.monitor;

**public** **final** **class** MonitorHeartBeat {

**boolean** keepProcessing\_ = **true**;

**public** MonitorHeartBeat() {

**try** {

Thread t = **new** Thread(**new** HealthCheck());

t.start();

}

**catch** (Exception e) {

}

}

**private** **class** HealthCheck **implements** Runnable {

**public** HealthCheck() {

}

@SuppressWarnings("unchecked")

**public** **void** run() {

**do** {

//your health check codes

//For example you make some web calls from this application

//to a remote server and get some information from that server

//now you can put a web call every after 5 minutes from here to check whether remote server

//is up and running or not if not you can send mail to concern team that the remove server is down

**try** {

System.***out***.println("I will add my health check code here");

Thread.*sleep*(10000);

}**catch**(Exception e) {

e.printStackTrace();

}

}**while** (keepProcessing\_);

}

}

**private** **final** **synchronized** **void** stopProcessing() {

keepProcessing\_ = **false**;

}

**public** **void** destroy() {

//This method need to be called from servlet

stopProcessing();

}

}

Now create package ‘com.we.system.startup’ and add below servlet ‘LoadOnStartUp’ into your project:-

**package** com.we.system.startup;

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**public** **class** LoadOnStartUp **extends** HttpServlet

{

**private** **static** com.web.system.monitor.MonitorHeartBeat *heartBeatMonitor* = **null**;

**private** HttpServlet mServlet;

**public** **void** init(ServletConfig servletConfig) **throws** ServletException

{

System.***out***.println("In LoadOnStartUp.init()");

//super.init(servletConfig);

//mServlet.init(servletConfig);

**new** AnnotationConfigApplicationContext(com.web.common.config.InitConfig.**class**);

*heartBeatMonitor* = **new** com.web.system.monitor.MonitorHeartBeat();// we created Heart beat monitor thread here

}

**public** **void** destroy()

{

//mServlet.destroy();

*heartBeatMonitor*.destroy();// we are stopping the thread here.

}

}

Provide reference to this servlet in web.xml file like below:-

<servlet>

<servlet-name>StartUpServlet</servlet-name>

<servlet-class>com.we.system.startup.LoadOnStartUp</servlet-class>

<init-param>

<param-name>disable-address-updates</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<!-- Make this servlet 1 and previous one as 2 -->

</servlet>

What this servlet will do? It’s init method will be called when you start this web service application in any of the web server (example IBM websphere) and it will instantiate the thread object from java class MonitorHeartBeat, which will keep checking the status of remote server whether it is up or down. This is an example only and hence I did not put any server check service in it. You can add threads as many as you want as per your requirement. Also from the init method it will load **class** InitConfig which is a configuration class I am introducing in this part.

In my previous project/article I did not use any log file and any application config file. Here I will introduce common configuration class to get all the config information for this application. Now I will add below class. This class will load all the configuration information from a .properties(config) file into a java class object which we are going to use in different section/places in the application.

 Create a new package call ‘com.web.common.config ‘ and add below ‘InitConfig’ class:-

**package** com.web.common.config;

**import** org.apache.log4j.BasicConfigurator;

**import** org.apache.log4j.PropertyConfigurator;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.PropertySource;

**import** org.springframework.core.env.Environment;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** javax.annotation.PostConstruct;

@Configuration

@PropertySource("file:C:\\Projects\\RESTService1\\mywebconfig.properties")

//you need to change this path and file name in your actual project

**public** **class** InitConfig {

@Autowired

**private** Environment env;

@PostConstruct

**public** **void** init(){

initConfig();

System.***out***.println("Initialized");

}

@Bean

**public** **static** ApplicationConfig applicationConfig(){

**return** ApplicationConfig.*getInstance*();

}

**public** InitConfig(){

}

**public** **void** initConfig() {

System.***out***.println("initConfig - CommonConfig");

String logPropertyFilePath = env.getProperty("LOG\_PROPERTY\_FILE\_PATH");

ApplicationConfig ac = ApplicationConfig.*getInstance*();

ac.setKEYSTOREPATH(env.getProperty("KEYSTOREPATH"));

ac.setTRUSTSTOREPATH(env.getProperty("TRUSTSTOREPATH"));

ac.setKEYSTOREPW(env.getProperty("KEYSTOREPW"));

ac.setTRUSTSTOREPW(env.getProperty("TRUSTSTOREPW"));

ac.setKEYPASS(env.getProperty("KEYPASS"));

ac.setHTTPS\_SERV\_URL(env.getProperty("HTTPS\_SERV\_URL"));

ac.setKeystoreType(env.getProperty("keystoreType"));

ac.setTrustAllCertificate(env.getProperty("trustAllCertificate"));

ac.setKeymanageralgorithm(env.getProperty("keymanageralgorithm"));

ac.setREGEX(env.getProperty("regex"));

**if** (logPropertyFilePath != **null**) {

System.***out***.println("Loading log properties file: "

+ logPropertyFilePath);

PropertyConfigurator.*configure*(logPropertyFilePath);

} **else** {

System.***out***.println("Called BasicConfigurator.configure()");

BasicConfigurator.*configure*();

}

}

}

And now I am going to add ApplicationConfig class. This class will hold all the configuration information, here I am using very minimal configuration information, you can add as much you want:-

**package** com.web.common.config;

**public** **final** **class** ApplicationConfig {

**private** String KEYSTOREPATH = **null**;

**private** String TRUSTSTOREPATH = **null**;

**private** String KEYSTOREPW = **null**;

**private** String TRUSTSTOREPW = **null**;

**private** String KEYPASS = **null**;

**private** String HTTPS\_SERV\_URL = **null**;

**private** String trustAllCertificate = "false";// DEFAULT VALUE

**private** String keystoreType = "JKS";// DEFAULT VALUE

**private** String regex = **null**;

**private** String keymanageralgorithm = **null**;

**private** **int** mqreadinterval = 1;

**private** **int** httpsfialureinterval = 5;

**private** **int** prodissueinterval = 1;

**private** **static** ApplicationConfig *myinstance* = **null**;

**public** **static** ApplicationConfig getInstance() {

System.***out***.println("in ApplicationConfig getInstance");

**if** (*myinstance* == **null**) {

*myinstance* = **new** ApplicationConfig();

}

**return** *myinstance*;

}

**private** ApplicationConfig() {

}

**public** String getKEYSTOREPATH() {

**return** KEYSTOREPATH;

}

**public** **void** setKEYSTOREPATH(String kEYSTOREPATH) {

KEYSTOREPATH = kEYSTOREPATH;

}

**public** String getTRUSTSTOREPATH() {

**return** TRUSTSTOREPATH;

}

**public** **void** setTRUSTSTOREPATH(String tRUSTSTOREPATH) {

TRUSTSTOREPATH = tRUSTSTOREPATH;

}

**public** String getKEYSTOREPW() {

**return** KEYSTOREPW;

}

**public** **void** setKEYSTOREPW(String kEYSTOREPW) {

KEYSTOREPW = kEYSTOREPW;

}

**public** String getTRUSTSTOREPW() {

**return** TRUSTSTOREPW;

}

**public** **void** setTRUSTSTOREPW(String tRUSTSTOREPW) {

TRUSTSTOREPW = tRUSTSTOREPW;

}

**public** String getKEYPASS() {

**return** KEYPASS;

}

**public** **void** setKEYPASS(String kEYPASS) {

KEYPASS = kEYPASS;

}

**public** String getHTTPS\_SERV\_URL() {

**return** HTTPS\_SERV\_URL;

}

**public** **void** setHTTPS\_SERV\_URL(String hTTPS\_SERV\_URL) {

HTTPS\_SERV\_URL = hTTPS\_SERV\_URL;

}

**public** String getTrustAllCertificate() {

**return** trustAllCertificate;

}

**public** **void** setTrustAllCertificate(String trustAllCertificate) {

**this**.trustAllCertificate = trustAllCertificate;

}

**public** String getKeystoreType() {

**return** keystoreType;

}

**public** **void** setKeystoreType(String keystoreType) {

**this**.keystoreType = keystoreType;

}

**public** String getKeymanageralgorithm() {

**return** keymanageralgorithm;

}

**public** **void** setKeymanageralgorithm(String keymanageralgorithm) {

**this**.keymanageralgorithm = keymanageralgorithm;

}

**public** **int** getMqreadinterval() {

**return** mqreadinterval;

}

**public** **void** setMqreadinterval(**int** mqreadinterval) {

**this**.mqreadinterval = mqreadinterval;

}

**public** **int** getHttpsfialureinterval() {

**return** httpsfialureinterval;

}

**public** **void** setHttpsfialureinterval(**int** httpsfialureinterval) {

**this**.httpsfialureinterval = httpsfialureinterval;

}

**public** **int** getProdissueinterval() {

**return** prodissueinterval;

}

**public** **void** setProdissueinterval(**int** prodissueinterval) {

**this**.prodissueinterval = prodissueinterval;

}

**public** **void** setREGEX(String regex) {

**this**.regex = regex;

}

**public** String getREGEX() {

**return** **this**.regex;

}

**public** **static** ApplicationConfig getMyinstance() {

**return** *myinstance*;

}

**public** **static** **void** setMyinstance(ApplicationConfig myinstance) {

ApplicationConfig.*myinstance* = myinstance;

}

}

Now actual part of this project. To make it ssl base project, I am going to introduce SSLConfig class to load a jks/p12  key and trust store including certificate to the application. You have to get the certificate from your target application and import it into your key store. You need to contact your target application server team for certificate. Either you can ask them for a certificate or you can provide your own certificate to the server team to install at their server end. First add the below SSLContextConfig class to your project:-

**package** com.web.common.ssl.config;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.security.KeyManagementException;

**import** java.security.KeyStore;

**import** java.security.KeyStoreException;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.SecureRandom;

**import** java.security.UnrecoverableKeyException;

**import** java.security.cert.CertificateException;

**import** javax.net.ssl.KeyManagerFactory;

**import** javax.net.ssl.SSLContext;

**import** javax.net.ssl.TrustManagerFactory;

**import** javax.net.ssl.TrustManager;

**import** javax.net.ssl.X509TrustManager;

**import** java.security.cert.X509Certificate;

**import** org.apache.log4j.Logger;

**import** com.web.common.config.ApplicationConfig;

**import** javax.net.ssl.KeyManager;

**public** **class** SSLContextConfig {

**private** **static** **final** Logger ***LOGGER*** = Logger.*getLogger*(SSLContextConfig.**class**);

**private** ApplicationConfig config\_ = ApplicationConfig.*getInstance*();

**private** TrustManager[] trustAllCerts = **null**;

**private** String keymanageralgorithm = **null**;

**public** SSLContext setupSslContext(){

SSLContext sslContext = **null**;

**boolean** trustall = **false**;

**try** {

String keyStorePath = config\_.getKEYSTOREPATH();

String trustStorePath = config\_.getTRUSTSTOREPATH();

String keyStorePw = config\_.getKEYSTOREPW();

String trustStorePw = config\_.getTRUSTSTOREPW();

String keyPass = config\_.getKEYPASS();

String trustAllCertificate = config\_.getTrustAllCertificate();

String keystoreType = config\_.getKeystoreType();

keymanageralgorithm = config\_.getKeymanageralgorithm();

trustAllCerts = **new** TrustManager[] {

**new** X509TrustManager() {

**public** java.security.cert.X509Certificate[] getAcceptedIssuers() {

**return** **null**;

}

**public** **void** checkClientTrusted(X509Certificate[] certs, String authType) { }

**public** **void** checkServerTrusted(X509Certificate[] certs, String authType) { }

}

};

**if**(trustAllCertificate.equalsIgnoreCase("True")){

trustall = **true**;

}

**if** (keystoreType.equalsIgnoreCase("JKS"))

sslContext = initializeSSLContext(keyStorePath, keyStorePw, trustStorePath, trustStorePw, keyPass,trustall);

**else**

sslContext = initializeSSLContextP12Cert(keyStorePath, keyStorePw, trustStorePath, trustStorePw, keyPass,trustall);

} **catch** (Exception exp) {

***LOGGER***.error("ConfigException exception occurred while reading the config file : " +exp.getMessage());

exp.printStackTrace();

}

**return** sslContext;

}

/\*\*

\*

\* **@param** keyStorePath

\* **@param** pwKeyStore

\* **@param** trustStorePath

\* **@param** pwTrustStore

\* **@param** keyPass

\* **@return**

\* **@throws** Exception

\*/

**private** SSLContext initializeSSLContext(**final** String keyStorePath, **final** String pwKeyStore, **final** String trustStorePath, **final** String pwTrustStore, **final** String keyPass, **final** **boolean** trustall) {

***LOGGER***.info(" In initializeSSLContext");

**char**[] keyStorePw = pwKeyStore.toCharArray();

**char**[] trustStorePw = pwTrustStore.toCharArray();

**char**[] keyPw = keyPass.toCharArray();

SecureRandom secureRandom = **new** SecureRandom();

secureRandom.nextInt();

KeyStore ks = **null**;

**try** {

ks = KeyStore.*getInstance*("JKS");

} **catch** (KeyStoreException exp) {

***LOGGER***.error("KeyStoreException exception occurred while reading the config file : " +exp.getMessage());

}

FileInputStream fis = **null**;

**try** {

**try** {

fis = **new** FileInputStream(keyStorePath);

} **catch** (FileNotFoundException exp) {

***LOGGER***.error("FileNotFoundException exception occurred " +exp.getMessage());

}

**try** {

ks.load(fis, keyStorePw);

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (CertificateException exp) {

***LOGGER***.error("CertificateException exception occurred " +exp.getMessage());

} **catch** (IOException exp) {

***LOGGER***.error("CertificateException exception occurred " +exp.getMessage());

}

} **finally** {

**if** (fis != **null**)

**try** {

fis.close();

} **catch** (IOException exp) {

***LOGGER***.error("IOException exception occurred " +exp.getMessage());

}

}

***LOGGER***.info("[initializeSSLContext] KMF keystorepw loaded.");

KeyManagerFactory kmf = **null**;

**try** {

kmf = KeyManagerFactory.*getInstance*(keymanageralgorithm);

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.error("IOException exception occurred " +exp.getMessage());

}

**try** {

kmf.init(ks, keyPw);

} **catch** (UnrecoverableKeyException exp) {

***LOGGER***.error("UnrecoverableKeyException exception occurred " +exp.getMessage());

} **catch** (KeyStoreException exp) {

***LOGGER***.error("KeyStoreException exception occurred " +exp.getMessage());

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

}

***LOGGER***.info("[initializeSSLContext] KMF init done.");

KeyStore ts = **null**;

**try** {

ts = KeyStore.*getInstance*("JKS");

} **catch** (KeyStoreException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

}

FileInputStream tfis = **null**;

SSLContext sslContext = **null**;

**try** {

tfis = **new** FileInputStream(trustStorePath);

ts.load(tfis, trustStorePw);

TrustManagerFactory tmf = TrustManagerFactory.*getInstance*(keymanageralgorithm);

tmf.init(ts);

***LOGGER***.info("[initializeSSLContext] Truststore initialized");

sslContext = SSLContext.*getInstance*("TLS");

**if**(trustall)

sslContext.init(kmf.getKeyManagers(), trustAllCerts,secureRandom);

**else**

sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers() ,secureRandom);

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (CertificateException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (IOException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (KeyStoreException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (KeyManagementException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **finally** {

**if** (tfis != **null**)

**try** {

tfis.close();

} **catch** (IOException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

}

}

**if**((sslContext == **null**)){

***LOGGER***.error("[initializeSSLContext] sslContext is null");

System.*exit*(-1);

}

**return** sslContext;

}

/\*\*

\*

\* **@param** keyStorePath

\* **@param** pwKeyStore

\* **@param** trustStorePath

\* **@param** pwTrustStore

\* **@param** keyPass

\* **@return**

\* **@throws** Exception

\*/

**private** SSLContext initializeSSLContextP12Cert(**final** String keyStorePath, **final** String pwKeyStore, **final** String trustStorePath, **final** String pwTrustStore, **final** String keyPass,**final** **boolean** trustall) {

***LOGGER***.info("In initializeSSLContextP12Cert");

SSLContext sslContext = **null**;

String keystore = keyStorePath;

String keystorepass = pwKeyStore;

String truststore = trustStorePath;

String truststorepass = pwTrustStore;

**try**{

KeyStore clientStore = KeyStore.*getInstance*("PKCS12");

clientStore.load(**new** FileInputStream(keystore), keystorepass.toCharArray());

KeyManagerFactory kmf = KeyManagerFactory.*getInstance*(keymanageralgorithm);

kmf.init(clientStore, keystorepass.toCharArray());

KeyManager[] kms = kmf.getKeyManagers();

KeyStore trustStore = KeyStore.*getInstance*("JKS");

trustStore.load(**new** FileInputStream(truststore), truststorepass.toCharArray());

TrustManagerFactory tmf = TrustManagerFactory.*getInstance*(keymanageralgorithm);

tmf.init(trustStore);

TrustManager[] tms = tmf.getTrustManagers();

sslContext = SSLContext.*getInstance*("TLS");

**if**(trustall)

sslContext.init(kms, trustAllCerts, **new** SecureRandom());

**else**

sslContext.init(kms, tms, **new** SecureRandom());

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.error("NoSuchAlgorithmException exception occurred " +exp.getMessage());

} **catch** (CertificateException exp) {

***LOGGER***.error("CertificateException exception occurred " +exp.getMessage());

} **catch** (IOException exp) {

***LOGGER***.error("IOException occurred while reading the key file " +exp.getMessage());

} **catch** (KeyStoreException exp) {

***LOGGER***.error("KeyStoreException exception occurred " +exp.getMessage());

} **catch** (KeyManagementException exp) {

***LOGGER***.error("KeyManagementException exception occurred " +exp.getMessage());

}**catch** (UnrecoverableKeyException exp) {

***LOGGER***.error("UnrecoverableKeyException exception occurred " +exp.getMessage());

}

**if**((sslContext == **null**)){

***LOGGER***.error("[initializeSSLContext] sslContext is null");

***LOGGER***.error("[initializeSSLContext] verify ssl config");

***LOGGER***.error("MyREST application exit with status code -1");

//System.exit(-1);

}

***LOGGER***.info("[initializeSSLContextP12Cert] Truststore and KeyStore initialized");

**return** sslContext;

}

}

I have designed the above class in such a way so that you can use both p12 keystore as well as jks key store. See the configuration file for details.

This class will load the certificate and generate a sslcontex object which we need to attach to a https connection object before making any remote https ssl based call. Please keep in mind if your target application do not want a SSL handshake you don’t really need any certificate. In this project we are going to make a GET request to Github server that does not requires a SSL handshake. Hence I have commented out assigning of sslContex object in https connection object. You can uncomment the codes and see what happens.

Now add below webserviceManager class into your project. This class will connect to remote webserver through https connection object and return the response back to client application that initiated the REST call to this server. This is the crucial part of the HTTPS communication. Here you might face lots of problems. Some of them are:-

1.       Handshake issue

2.       Various Certificate related issues

3.       URL related issues.

You have to carefully analyse each issue and fix it. Fixing of issues are not covered here.

But for the purpose of successful testing I have commented out the certificate attachment part in the code as I am going to make call to some open source remote server where you don’t need a certificate.

package com.web.webservice.manager;

import com.web.common.config.ApplicationConfig;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.StringWriter;

import java.io.Writer;

import java.net.URL;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLContext;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.UriInfo;

import javax.ws.rs.core.Response.Status;

import com.web.common.ssl.config.SSLContextConfig;

import org.apache.cxf.helpers.IOUtils;

import org.apache.log4j.Logger;

import org.codehaus.jackson.map.ObjectMapper;

public class WebServiceManager {

static int \_responseCode = -1;

static private ApplicationConfig config\_ = ApplicationConfig.getInstance();

private static final Logger LOGGER = Logger.getLogger(WebServiceManager.class);

static HttpsURLConnection connection\_ = null;

static SSLContext sslContext = null;

static {

//SSLContextConfig sslconfig = new SSLContextConfig();

//sslContext = sslconfig.setupSslContext();

LOGGER.info("Webservice is loading...");

}

//------------------------------------------------------------------

static public boolean setSSLConnection(UriInfo info) {

System.out.println("In setSSLConnection");

URL url = null;

//HttpsURLConnection.setDefaultSSLSocketFactory(sslContext

// .getSocketFactory());

try {

url = new URL(null, WebServiceManager.generateTargetURL(info),

new sun.net.www.protocol.https.Handler());

} catch (Exception e1) {

LOGGER.error("MalformedURLException occurred " + e1.getMessage());

}

try {

connection\_ = (HttpsURLConnection) url.openConnection();

//connection\_.setSSLSocketFactory(sslContext.getSocketFactory());

connection\_.setDoOutput( true );

connection\_.setRequestProperty( "Content-Type", "text/xml" );

connection\_.connect();

return true;

} catch (Exception e) {

LOGGER.error("Exception occurred while establishing connection to SSL server. Error :"

+ e.getMessage());

connection\_.disconnect();

connection\_ = null;

return false;

}

}

//----------------------------------------------------------------------------------

static public String generateTargetURL(UriInfo info) {

LOGGER.info("In generateTargetURL()");

String url = null;

String regex = config\_.getREGEX();

String sevcURL[] = info.getRequestUri().toString().split(regex);

String serviceURL = config\_.getHTTPS\_SERV\_URL();

url = serviceURL + sevcURL[1].replace("/"+"Github.json","").replace("/v/github/", "");

LOGGER.info("URL is : " + url);

return url;

}

static public Response sendGETmsg(UriInfo info) {

LOGGER.info("In sendGETmsg");

String response = null;

int retcode = 500;

if(!WebServiceManager.setSSLConnection(info)) {

response = "Failed to setSSLConnection ";

return Response.status(Status.INTERNAL\_SERVER\_ERROR).entity(response).type(MediaType.APPLICATION\_JSON).build();

}

try {

//connection\_.setRequestMethod("GET");

retcode = connection\_.getResponseCode();

System.out.println("In sendGETmsg2 :" + retcode);

if(retcode != 401 && retcode != 500)

response = IOUtils.toString(connection\_.getInputStream());

else

response="Failed to connect to remote server :";

}catch(Exception e) {

e.printStackTrace();

}

connection\_.disconnect();

return Response.status(retcode).entity(response).type(MediaType.APPLICATION\_JSON).build();

}

//Now send the message to the server:-

public static Response sendPOSTmsg(UriInfo info, Object sendObject) {

String response = null;

if(!WebServiceManager.setSSLConnection(info)) {

response = "Failed to setSSLConnection ";

return Response.status(Status.INTERNAL\_SERVER\_ERROR).entity(response).type(MediaType.APPLICATION\_JSON).build();

}

ObjectMapper mapper = new ObjectMapper();

try{

connection\_.setRequestMethod("POST");

Writer strWriter = new StringWriter();

mapper.writeValue(strWriter, sendObject);

String reqJSONData = strWriter.toString();

//Sending the request to Remote server

OutputStreamWriter writer = new OutputStreamWriter(connection\_.getOutputStream());

writer.write(reqJSONData);

writer.flush();

writer.close();

\_responseCode = connection\_.getResponseCode();

LOGGER.info("Response Code :" + \_responseCode);

// reading the response

InputStreamReader reader = new InputStreamReader(connection\_.getInputStream());

StringBuilder buf = new StringBuilder();

char[] cbuf = new char[ 2048 ];

int num;

while ( -1 != (num = reader.read( cbuf )))

{

buf.append( cbuf, 0, num );

}

response = buf.toString();

}catch(Exception e){

response = "<EXCEPTION>Exception occurred while sending message</EXCEPTION>";

e.printStackTrace();

}

connection\_.disconnect();

return Response.status(Status.OK).entity(response).type(MediaType.APPLICATION\_JSON).build();

}

}

Finally we have done with everything. Even though I have introduced the POST call here I am not providing any test result as I did not find any open source POST call available online.

Now right click on your project and select `Run` and select server. Once the webservice starts running successfully, do the below test.

In your SOAPI tool select File and then select “New REST Project” and put the below url and following the instruction provided in my previous project:-

http://localhost:9080/ RESTService1/rest/example/v/github/Github.json/repositories?sort=stars&q="Apps Script" stars:">=100"

Most of the necessary functionalities have been added in this REST SSL base web service project. You can even utilize this frame work for your company for business purpose. Please provide me your necessary input if anything is missing which I may add based on the advice in future release.

Thanks for reading this article.