**NOTE: This uses oracle webLogic server and glassFish.**

**Web Services Part 5 - How to create RESTful Web Services (Hands on using Eclipse and Jersey) BY:** [gontuseries](https://www.youtube.com/channel/UCA3RaMnfSmJsvv3TraOdipg)   
WATCH: <https://www.youtube.com/watch?v=n_agJFIB9bw>  
  
@0:00-1:01: Summary of what is to come.  
@1:01: In project explorer: RightClick > New > Dynamic Web Project  
@1:07: Name: RESTfulWS  
@1:15: Choose [New Runtime]  
@1:31: Presenter will be using: Oracle > Oracle WebLogic Server 10gR3  
We however, want to use TomCat. Hopefully wont be too hard to implement  
this tutorial when using a different server type.  
@1:37: Dialog: “Oracle WebLogic Server 10gR3: Define a WebLogic Runtime:”  
@1:40: Providing WebLogic home. (Where the server installation is I believe)  
@1:58: Click Finish.  
@2:01: Check: Add project to an EAR.

|  |  |
| --- | --- |
| An **EAR file** is a standard JAR **file** (and therefore a Zip **file**) with a .**ear** extension, with one or more entries representing the modules of the application, and a metadata directory called META-INF which contains one or more deployment descriptors.   (WIKIPEDIA) | **.war vs .ear:** http://stackoverflow.com/questions/1594667/war-vs-ear-file In J2EE application modules are packaged as EAR, JAR and WAR based on their functionality  JAR: EJB modules which contains enterprise java beans class files and EJB deployment descriptor are packed as JAR files with .jar extenstion  WAR: Web modules which contains Servlet class files,JSP FIles,supporting files, GIF and HTML files are packaged as JAR file with .war (web archive) extension  EAR: All above files (.jar and .war) are packaged as JAR file with .ear (enterprise archive) extension and deployed into Application Server. |

@2:04: Hit Next.  
@2:07: Hit Next.  
@2:08: Hit Finish.  
@2:13-2:18: This wizard will create the required folder structures for RESTFul web services.  
@2:19-2:26: We are creating this restufl web service using Jersey, so we will need Jersey jars.  
<http://jersey.java.net>  
@2:26: JMINNOTE: This is a glassfish thing by the looks of the site. Maybe I should stop now and  
look at what other options there are? Maybe finish transcribing, because there are bound to be analogous systems.  
@2:36: Click on “zip of Jersey”  
@2:49: Copying everything out of “lib” folder of extracted zip file.  
@2:56: Pasting them into project under:  
<prjRoot>/WebContent/WEB-INF/lib  
@3:06: Now create a university resource.  
\*Right Clicks on “Java Resources” folder in Project Explorer.  
@3:10 NEW --> Class  
@3:12 Package Name: com.gontuseries.university  
@3:20-3:29: Give a name to the class: “UniversityRESTWS”  
@3:30: Click Finish.  
@3:35-3:38: Lets modify the default template class.  
@3:48-3:56: This annotation specifies that this restfull web service is exposing a university resource.  
@3:56-4:00: Which can be accessed by the client using this URI:  
<http://localhost:7001/RESTfulWS/rest/university>

UniversityRESTWS.java://///////////////////////  
package com.gontuseries.university;  
import javax.ws.rs.Path;  
@Path("/university")  
public class UniversityRESTWS{} //currently empty.  
//////////////////////////////////////////////////  
@4:15: About to explain the university class to save some time. (or was that..not?)  
@4:17: Two GET methods.  
@4:18-4:20: And one PUT method.  
@4:21-4:26: If we “cline?” AH:  
**“If the client wants the response in HTML format.”**@4:26: Then this method will get executed.  
@4:29: And if the client wants the response in plain text.  
@4:33: Then this method will get executed.  
@4:36-4:45: (At Direct?) Produces Annotation tells this method will produce the response only in this specified format.  
@4:46-4:54: If the client wants to update the student information present in the university restful webservices.  
@4:54-5:00: He can do so by calling the @PUT method of the university resource.  
\*Pointing to updateUniversityInfo(...) function. \*  
@4:59-5:07: The client can pass some student’s specific to the server using this URL.  
<http://localhost:7001/RESTfulWS/rest/university/24>

@5:08 – 5:13:In this case, client is passing student’s roll number to the method.  
\*Pointing at the version of updateUniversityInfo with the @Path annotation.\*  
  
@5:13-5:16:And the method would recieve this parameter.  
@5:16-5:21: Using **[indiscernible]** path notation just before the method.  
@5:22-5:28: And update university info method would retrieve this student roll number.  
@5:28-5:31 From the URI using this syntax: @PathParam(“studentRollNo”)  
@5:35-5:43: Then you can use student roll number to update that students specific information in the database.  
@5:44-5:48 And then you can return back the success method to the client.  
@5:48-5:51: Now our university resource is ready.  
@5:52-5:57: Client can access. Two get methods and one put method in this university resource.  
@5:57-6:03: Lets modify the default web.xml created by eclipse web wizard.  
@6:09: Opens web.xml within <prj>/WebContent/WEB-INF/web.xml  
@6:13-6:16: This is the default web.xml created by eclipse wizard.  
@6:18-6:22: Just replace the contents of this file with this one.

|  |
| --- |
| //delete everything inside <web-app> and put the following code:  <servlet>  <servlet-name>Jersey REST Service</servlet-name>  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>  <init-param>  <param-name>com.sun.jersey.config.property.packages</param-name>  <param-value>com.gontuseries.university</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>Jersey REST Service</servlet-name>  <url-pattern>/rest/\*</url-pattern>  </servlet-mapping> |

@6:27: “/rest/\*” It’s the url pattern used to build URI – see how to use it while testing after some time in this demo.  
@6:27-6:32: Your RESTFul web services is now ready to build and deploy.  
@6:38: <prjRoot> + RIGHTCLICK --> Run As --> Run On Server (Alt+SHIFT+X)  
@6:41-6:43: Choose the server.  
@6:44: Click Next.  
@6:47-6:52: If you have already created the domain of the server, then provide the path here.  
\*Points to “Domain directory” text field within “Specify a WebLogic domain directory.  
@6:52-6:58:In this example I am going to create a new instance of the server that I am using.  
@7:00-7:08 :Providing the name and (optionally) location of the server if we want REST domain to be created.  
Name: RESTDemo  
Location: D:\RESTDemoDomain  
@7:12: Click on finish.  
@7:13-7:16: And this will create the domain of the server.  
@7:15-7:16: Clicks FINISH.  
@7:17-7:21: This will deploy the restful web services on the server.  
@7:25: ERROR 403—FORBIDDEN  
http://localhost:7001/RESTfulWS/  
@7:25-7:28: Now your restful web service is up and running.  
@7:28-7:32:You can test restful web services directly on the web browser.  
@7:36: Types: <http://localhost:7001/RESTfulWS/rest/> into browser.  
http://localhost:7001/[context root of the application]/ [<url-pattern> defined in web.xml ] / resource name  
@7:39: <http://localhost:7001/RESTfulWS/rest/university>  
I now see: “NAME- India University” in the web browser.  
@7:40-7:47: By default the browser calls up the get method of the university restful web services.  
@7:47-7:52: And requests for the HTML representation of the resource.  
@7:52-7:58: You may use JAX\_RS APIs to write a class and test RESTful web services.  
@7:58-8:11: Or you can test it using standard tools like “restclient-ui”.  
Download: http://code.google.com/p/rest-client/