**Restfull API:**

If you are in this web world or web development world then definitely we will have to go through Rest API. It does not matter whether we are working with Java, Dot net, PHP, Ruby etc.

**What is an API?**

* It stands for Application Programming Interface (API). It is a set of Interfaces, methods, classes (codes) written by someone else and we can use these in our application development.
* For example Google signup written by google but we can used this API for login to our application. We don’t need to bother about what they have written, how they are doing authentication or authorization etc.
* Similarly Facebook signup written by Facebook but we can use this Facebook for login to our application.
* Most of time we have to use API Key so that the Google or Facebook can authenticate and can track if we are doing any illegal activity with the key.
* API keys identifies our app and our request.
* It is the part of request services.
* It can identify the user can log their request.
* It can accept/ reject our request based on API Key.
* It can authorize and identify our request and check whether we are eligible to get Data from API.

**Q**:-Now the question comes what exactly REST means?

* Let’s take an example: Let’s say we have to develop one app for taking weather report, then generally what we will do we will capture the weather according to time and location.
* Now for capturing these data according to time and location we need to put sensor across the world or need to launch satellite which send the data to our sensor and we can get these data from the sensor. But for developing a small app obviously we are not going to these stupid things.
* For getting those data we will just one request to the server which are getting these data from the satellite or somewhere else. This is nothing but we are getting service from the server by requesting with some questions like time and location.
* Now the question comes how we generate data. Since here we are talking about java so here we will use Servlet who will generate the data and client will the send the request to servlet and it will give the data to client.
* Now the question what servlet will return in response? Generally when we any client (like browser) any request to the servlet then in response we get an Html page. Now wait here we don’t need any HTML page because I need only data (because if we get html then we need to parse it and then we will get data).
* So we will design the client request such that it will send only data. This data could be either in the form XML or JSON format. Now why only xml or json.
* Because when get some data from the server then it should be structured (Tree and parent-child based structure) and XML or JSON gives the data in structured format so that we can understand.
* Now a days most of the people used JSON format because it is easier to understand in compare to XML.

|  |
| --- |
|  |

* Hence finally servlet will return the data either in XML or JSON format and hence our job is done.
* But one more problem using this servlet is, as when we use servlet then we have to write lot of things like overriding service methods like (doGet or doPost etc).
* So to replace the above things we have one simple solution and that solution is nothing but REST API that we can use in any technology (like Java, Dot net, and PHP) etc.
* So REST says if you want some data then just simply create an object from the server side and return the object(having values either in XML or JSON format) to client. So we have an object and we want to send a state of an object in the form of XML or JSON format i.e. it we are transferring the state of object and representing it in a particular format (XML or JSON) that is why it called as **RepresentationalStateTransfer.**
* Generally in all the application we used to do CRUD (CREATE, READ, UPDATE and DELETE) operation with the resource.
* For doing these CRUD operation we use HTTP methods (POST for Create, GET for read, PUT for update, DELETE for delete resource).

**Implementation of REST API:**

Generally we have different approach to develop RESTFull API but mostly we use two approaches

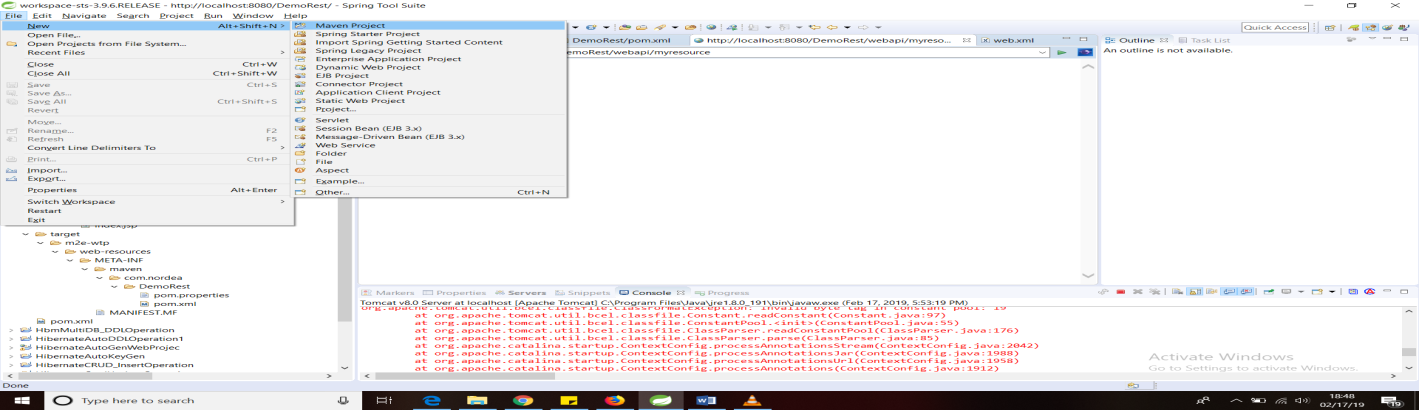
1. Jersey and 2- Spring implementation of Jersey

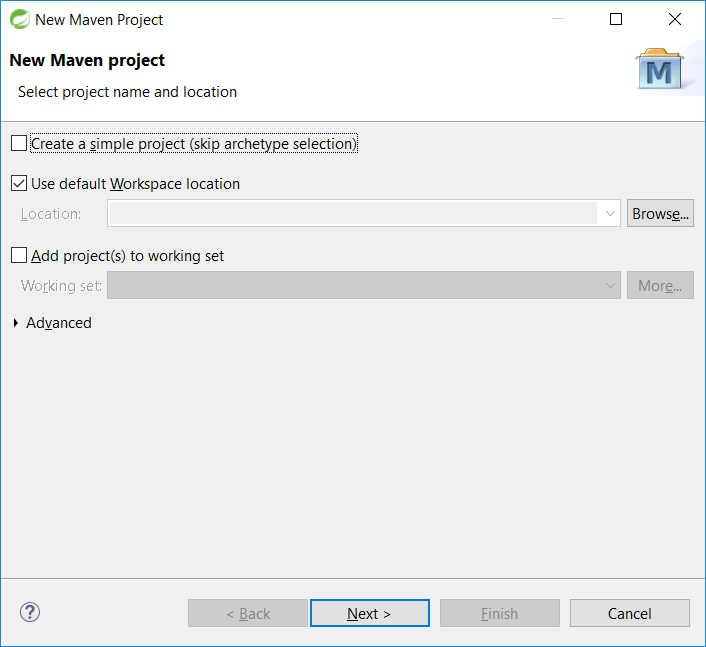
**Note: Difference between JAX-RS and Jersey.**

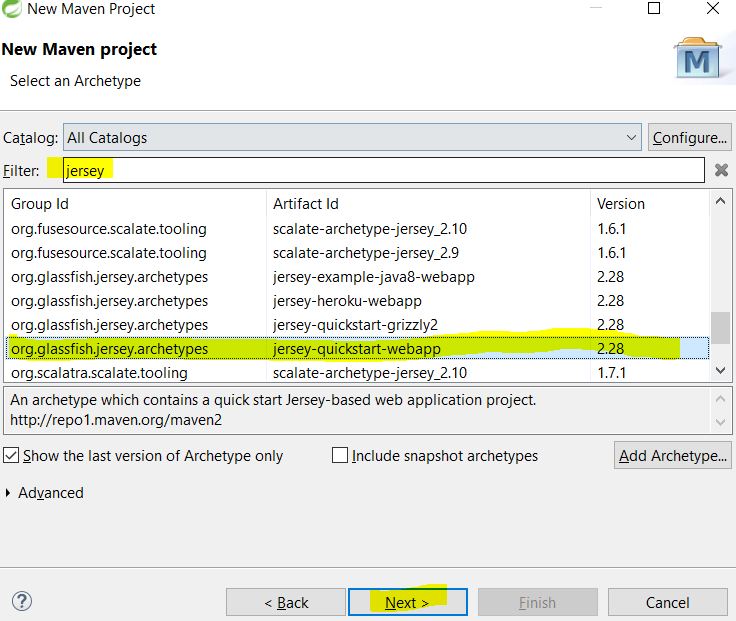
JAX-RS is a specification and jersey is an implementation of JAX-RS. For example JPA is a specification but hibernate is an implementation of Jersey. Apart from Jersey we have some other implementation of JAX-RS like RestEasy and IBM implementation of JAX-RS etc.

**First RestApi project creation.**

1. Create a maven project from eclipse: File-🡪 New-🡪 Maven Poject

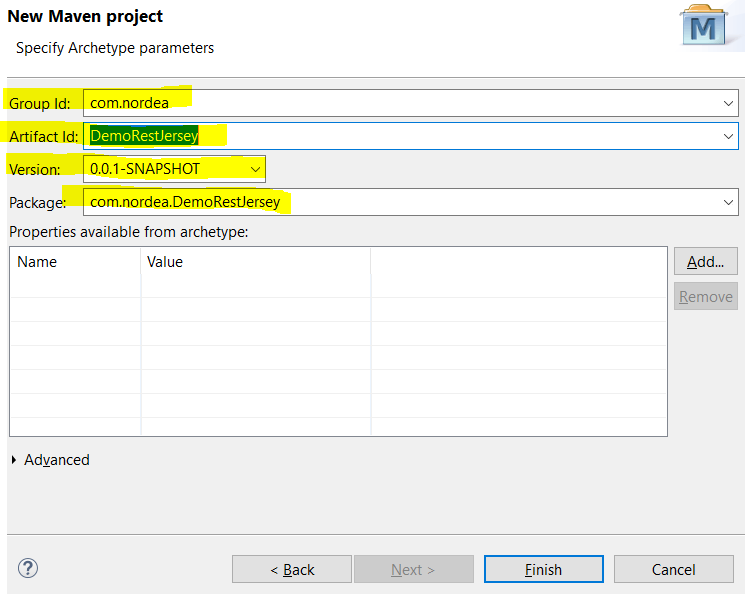




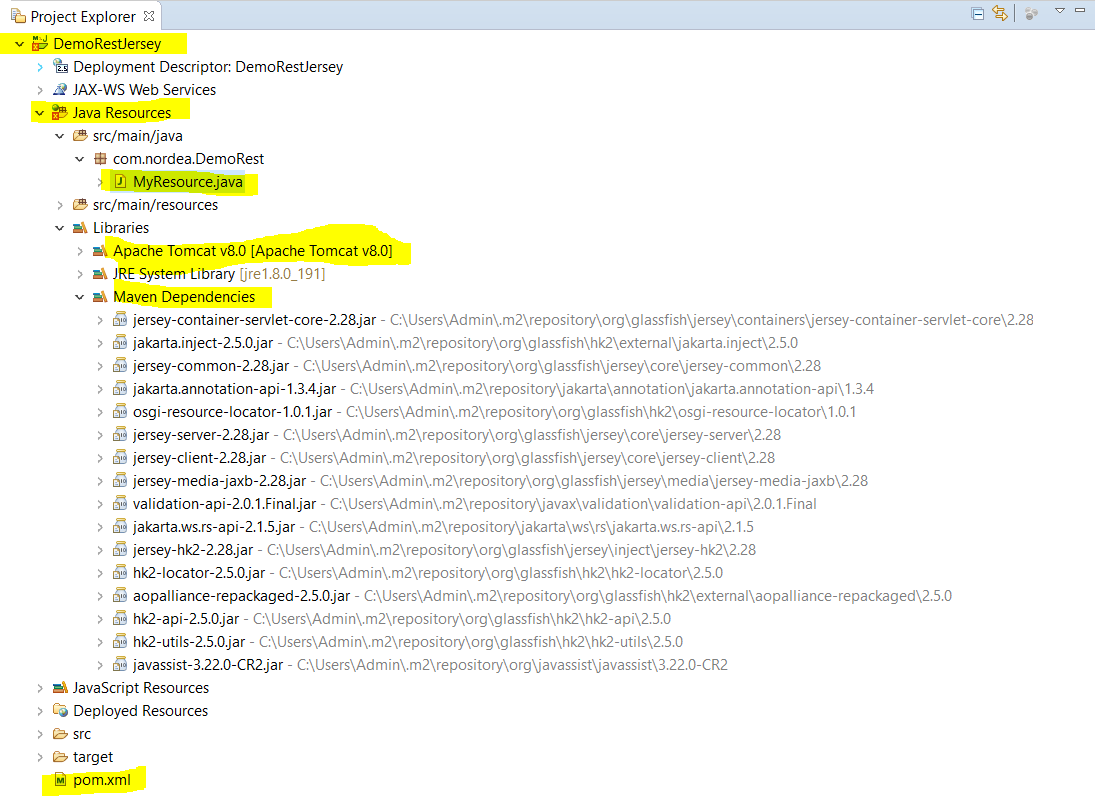
1. **Click Next and use default :**

Here in filter just type Jersey to get jersey-quickstart-webapparchtype. Select jersey-quickstart-webapp and clikc next

1. Mention Group Id: Artifact Id: Version: Package as shown in the below image.

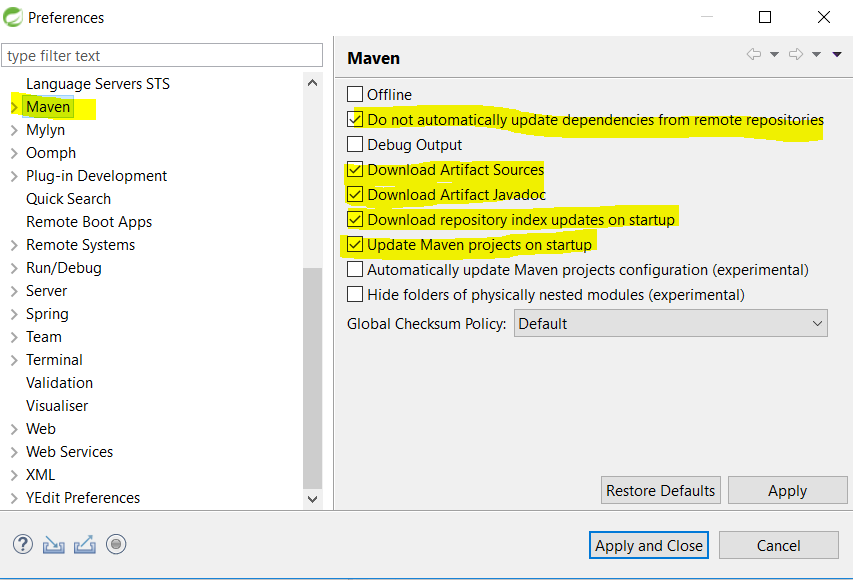


1. Click finish
2. As soon as we click on finish. It start creating project as well as it start downloading all the jars required to run Jersey application. Finally the project looks like given below.

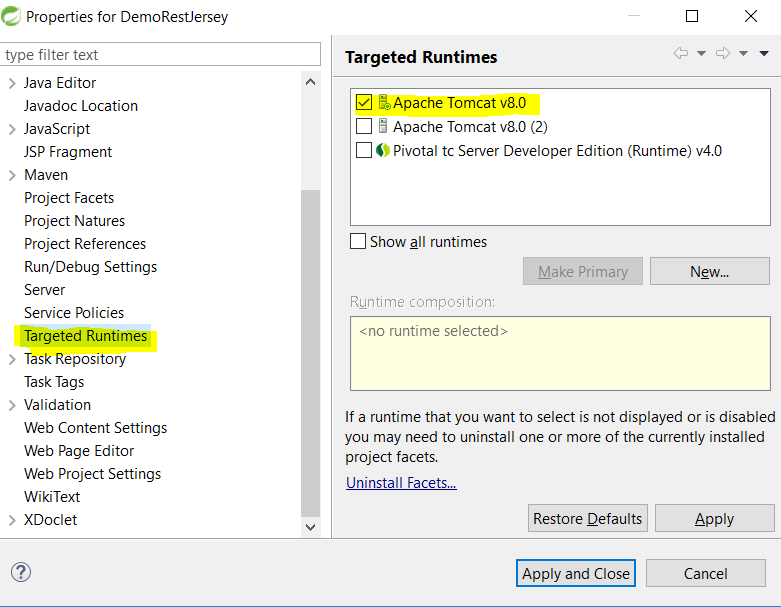


Point to be noted:

1. It got a well-defined structure of web application with root java class [MyResource.java].
2. Maven dependencies with all dependent jars.
3. Pom.xml. Latter we will discuss about this.
4. We need to check and do some configuration if we are getting any build issue.
5. Go Window-🡪 Preferences and we need to check some check box as mentioned in the below screen.



1. Right click on the project --- > properties -🡪 Target runtime --- > Select target runtime Tomcat server8.



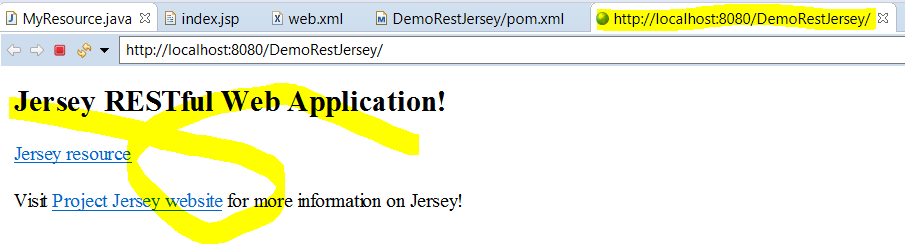
1. Go to project facet and change the java version to 1.8 from 1.7
2. Now let’s come to index.jsp

|  |
| --- |
| <html>  <body>  <h2>Jersey RESTful Web Application!</h2>  <p><ahref=*"webapi/myresource"*>Jersey resource</a>  <p>Visit <ahref=*"http://jersey.java.net"*>Project Jersey website</a>  for more information on Jersey!  </body>  </html> |

Here [href=*"webapi/myresource"] is the root url to run project.*

9 Now run the application: Run on server: When we do this then we get the fallowing url and webpage.

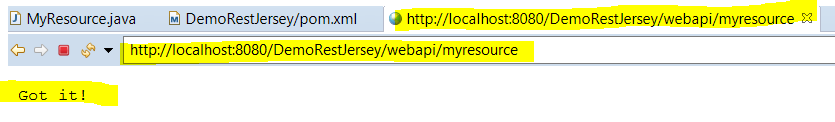
[**http://localhost:8080/DemoRestJersey/**](http://localhost:8080/DemoRestJersey/)



**Hence to get proper output we need to append** *webapi/myresource in the* [**http://localhost:8080/DemoRestJersey/**](http://localhost:8080/DemoRestJersey/)

**Hence final URL will be:** [**http://localhost:8080/DemoRestJersey/**](http://localhost:8080/DemoRestJersey/) **webapi/myresource**

**Now if we execute the above URL then we will get proper result**



**Let’s discuss about Web.xml and MyResource root class**

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <!-- This web.xml file is not required when using Servlet 3.0 container,  see implementation details http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->  <web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*version=*"2.5"*>  <servlet>  <servlet-name>Jersey Web Application</servlet-name>  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>  <init-param>  <param-name>jersey.config.server.provider.packages</param-name>  <param-value>com.nordea.DemoRest</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>Jersey Web Application</servlet-name>  <url-pattern>/webapi/\*</url-pattern>  </servlet-mapping>  </web-app> |
| **package**com.nordea.DemoRest;  **import**javax.ws.rs.GET;  **import**javax.ws.rs.Path;  **import**javax.ws.rs.Produces;  **import**javax.ws.rs.core.MediaType;  /\*\*  \* Root resource (exposed at "myresource" path)  \*/  @Path("myresource")  **publicclass** MyResource {  /\*\*  \* Method handling HTTP GET requests. The returned object will be sent  \* to the client as "text/plain" media type.  \*  \* **@return** String that will be returned as a text/plain response.  \*/  @GET  @Produces(MediaType.***TEXT\_PLAIN***)  **public** String getIt() {  **return**"Got it!";  }  } |

1. [**http://localhost:8080/DemoRestJersey/**](http://localhost:8080/DemoRestJersey/)**webapi/myresource**

* **As soon as we put this URL on browser it Tomcat container first goes to web.xml and it sends the request to <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>**ServletContainer class which try to search path**@Path ("myresource")**in the application
* Now it goes to MyResource class and calls **getit()**method as here we have only one @Get method then it will call this method as by default only Get method gets executed.
* In the get method the MediaType is TEXT\_PLAIN. So it will return simple text to browser.

**Let’s create our own resource:**

|  |  |
| --- | --- |
| **package** com.nordea.employee;  **import** javax.xml.bind.annotation.XmlRootElement;  @XmlRootElement  **publicclass** Employee {    **private**StringempName;  **private** String empId;  **privateint**age;  **private** String address;  **public** String getEmpName() {  **return**empName;  }  **publicvoid**setEmpName(String empName) {  **this**.empName = empName;  }  **public** String getEmpId() {  **return**empId;  }  **publicvoid**setEmpId(String empId) {  **this**.empId = empId;  }  **publicint**getAge() {  **return**age;  }  **publicvoid**setAge(**int**age) {  **this**.age = age;  }  **public** String getAddress() {  **return**address;  }  **publicvoid**setAddress(String address) {  **this**.address = address;  }  } | **package** com.nordea.employee;  **import**javax.ws.rs.GET;  **import**javax.ws.rs.Path;  **import**javax.ws.rs.Produces;  **import**javax.ws.rs.core.MediaType;  @Path("employee")  **publicclass**GetEmpResource {    @GET  @Produces(MediaType.***APPLICATION\_XML***)  **public** Employee getEmployee() {    System.***out***.println("Inside getEmployee....");  Employee emp = **new** Employee();  emp.setEmpName("Arun");  emp.setEmpId("G82234");  emp.setAge(30);  emp.setAddress("Bangalore");  **return**emp;  }  } |

[**http://localhost:8080/DemoRestJersey/webapi/employee**](http://localhost:8080/DemoRestJersey/webapi/employee)

**Let’s say if I want to return a list of employee.**

* In this case inside GetEmpResource the existing method [**public** Employee getEmployee()] will replaced with list [**public** List<Employee>getEmployee()]
* Next we will add another object [Employee emp1 = **new** Employee();]
* Then using emp1 we will set emp.setEmpName("Arun");emp.setEmpId("G82234");emp.setAge(30);
* emp.setAddress("Bangalore");
* Finally we will create one arraylist and will add these object in the Arraylist and will return same arraylist
* Hence the above code will be looks like given below

|  |
| --- |
| @GET  @Produces(MediaType.***APPLICATION\_XML***)  **public** List<Employee>getEmployees(){  Employee emp = **new** Employee();  emp.setEmpName("Arun");  emp.setEmpId("G82234");  emp.setAge(30);  emp.setAddress("Bangalore");  //------------------------------  Employee emp2 = **new** Employee();  emp2.setEmpName("Arun");  emp2.setEmpId("G82234");  emp2.setAge(30);  emp2.setAddress("Bangalore");    List<Employee>empList = Arrays.*asList*(emp,emp2);  **return**empList;  }  } |

Output:

Mock Repository