WEB SERVICES

In web service

we will see the introduction of webservices in java and some jargons of web services.

Webservices in java are used everywhere nowadays. When human interacts with any web page, it involves request and response via HTML. When you interact with the webpage, browser sends a request and then renders response and shows in form of HTML. Similarly, web services also involve request and response but in the form of XML or JSON or plain text. It generally used for other applications or programs to consume and make use of information.

Forexample::
You are creating a website which shows weather information of important cities in the world. You can actually consume already exposed web services and get the data for the cities. You will get the response in form of XML or JSON, you can parse it and show it on your website.

Let's go to more formal definition now:
Web service is a way of communication that allows interoperability between different applications on different platforms, for example, a Java based application on Windows can communicate with a .Net based one on Linux. The communication can be done through a set of XML messages over HTTP protocol.Web services are browsers and operating system independent service, which means it can run on any browser without the need of making any changes. Web Services take Web-applications to the Next Level.

The World Wide Web Consortium (W3C) has defined the web services. According to W3C, "Web Services are the message-based design frequently found on the Web and in enterprise software. The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL, and others."
Let's say, you are a Java developer and you can publish your functions on internet or LAN through java web service so any other developer(let's say .Net developer) can access your function.

You can go through web services interview questions for interview questions on webservices in java.

Why you need to learn web services:

Reuse already developed(old) functionality into new software:

Let's understand with the very simple example.Let's say you are developing a finance software for a company on java and you have old .net software which manages salary of employees.So rather than developing new software for employee part, you can use old software and for other parts like infrastructure, you can develop your own functionalities.

Usability :

Web Services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web Services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.

Interoperability :

This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their solutions.Web Services also let developers use their preferred programming languages. In addition, thanks to the use of standards-based communications methods, Web Services are virtually platform-independent.

Loosely Coupled:

Each service exists independently of the other services that make up the application. Individual pieces of the application to be modified without impacting unrelated areas.

Ease of Integration:

Data is isolated between applications creating 'silos'. Web Services act as glue between these and enable easier communications within and across organizations.

Deployability :

Web Services are deployed over standard Internet technologies. This makes it possible to deploy Web Services even over the firewall to servers running on the Internet on the other side of the globe. Also thanks to the use of proven community standards, underlying security (such as SSL) is already built-in.

Some jargons used in Webservices in java:

Simple Object Access Protocol(SOAP):

SOAP is a protocol specification for exchanging structured information in the implementation of Web services in computer networks. It relies on XML as its message format.

Web Service Description Language(WSDL):

WSDL stands for Web Service Description Language. It is an XML file that describes
the technical details of how to implement a web service, more specifically the URI,
port, method names, arguments, and data types. Since WSDL is XML, it is both human-readable and machine-consumable, which aids in the ability to call and bind to
services dynamically.

Elements of WSDL are:

Description:

It is the root element of a WSDL 2.0 file. It usually contains a set of namespace declarations which are used throughout the WSDL file.

Types:

The WSDL types element describes the data types used by your web service.Data types are usually specified by XML schema.It can be described in any language as long as your web services API supports it.

Binding:

The WSDL binding element describes how your web service is bound to a protocol. In other words, how your web service is accessible. To be accessible, the web service must be reachable using some network protocol. This is called “binding” the web service to the protocol.

Interface:

The WSDL interface element describes the operations supported by your web service.It is similar to methods in programming language.The client can only call one operation per request.

Service:

It describes the endpoint of your web service. In other words, the address where the web service can be reached.

Endpoint:

The endpoint element describes the address of the web service. The **endpoint binding** attribute describes what binding element this endpoint uses.i.e. protocol with which you will access web service. The **address attribute** describes the URI at which you can access the service.

Message:

The message element describes the data being exchanged between the Web service providers and consumers.

Sample WSDL file:

```
1
2
```

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://webservices.javapostsforlearning.arpit.org"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://webservices.javapostsforlearning.arpit.org"
xmlns:intf="http://webservices.javapostsforlearning.arpit.org"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace="http://webservices.javapostsforlearning.arpit.org"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="sayHelloWorld">
    <complexType>
     <sequence>
      <element name="name" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="sayHelloWorldResponse">
    <complexType>
     <sequence>
      <element name="sayHelloWorldReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
  <wsdl:message name="sayHelloWorldRequest">
    <wsdl:part element="impl:sayHelloWorld" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="sayHelloWorldResponse">
    <wsdl:part element="impl:sayHelloWorldResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="HelloWorld">
    <wsdl:operation name="sayHelloWorld">
      <wsdl:input message="impl:sayHelloWorldRequest" name="sayHelloWorldRequest"/>
      <wsdl:output message="impl:sayHelloWorldResponse" name="sayHelloWorldResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloWorldSoapBinding" type="impl:HelloWorld">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sayHelloWorld">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="sayHelloWorldRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="sayHelloWorldResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloWorldService">
    <wsdl:port binding="impl:HelloWorldSoapBinding" name="HelloWorld">
      <wsdlsoap:address location="http://localhost:8080/SimpleSOAPExample/services/HelloWorld"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Universal Description, Discovery and Integration(UDDI):

UDDI stands for Universal Description, Discovery, and Integration.It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery

Web service design approaches:

Contract last or Bottom-up approach:

When using contract last approach, you first write your Java code then you create web service contract(WSDL).There are various kinds of tools which can generate WSDL on the basis of Java code.

Contract first or Top Down Approach:

It is reverse of contract first.Here you first define web service contract.You define all the elements of WSDL first then after that you create your java logic.

Types of webservices in java:

There are mainly two types of web services.

- REST
- SOAP

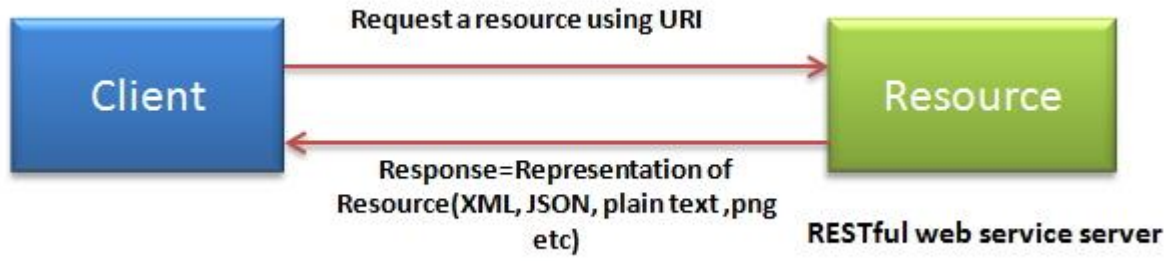
You can read about differences and usage of REST and SOAP web services. That's all about the webservices in java.

RESTful web service::

REST is an architectural style which was brought in by Roy Fielding in 2000 in his doctoral thesis.In the web services terms, REpresentational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web service clients that want to use these resources access via globally defined set of remote methods that describe the action to be performed on the resource.

It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.

In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP. The response from server is considered as the representation of the resources. This representation can be generated from one resource or more number of resources.



REST allows that resources have different representations, e.g.xml, json etc. The rest client can ask for specific representation via the HTTP protocol.

HTTP methods :

RESTful web services use HTTP protocol methods for the operations they perform.Methods are:

- **GET**:It defines a reading access of the resource without side-effects.This operation is idempotent i.e.they can be applied multiple times without changing the result
- **PUT** : It is generally used for updating resouce.It must also be idempotent.
- **DELETE** : It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
- **POST** :It is used for creating a new resource. It is not idempotent.

Idempotent means result of multiple successful request will not change state of resource after initial application

For example :

Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.

Post is not idempotent method because when you use post to create resource , it will keep creating resource for each new request, so result of multiple successful request will not be same.

Features of RESTful web services:

Resource identification through URI:Resources are identified by their URIs (typically links on internet). So, a client can directly access a RESTful Web Services using the URIs of the resources (same as you put a website address in the browser's address bar and get some representation as response).

Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.

Client-Server: A clear separation concerns is the reason behind this constraint. Separating concerns between the Client and Server helps improve portability in the Client and Scalability of the server components.

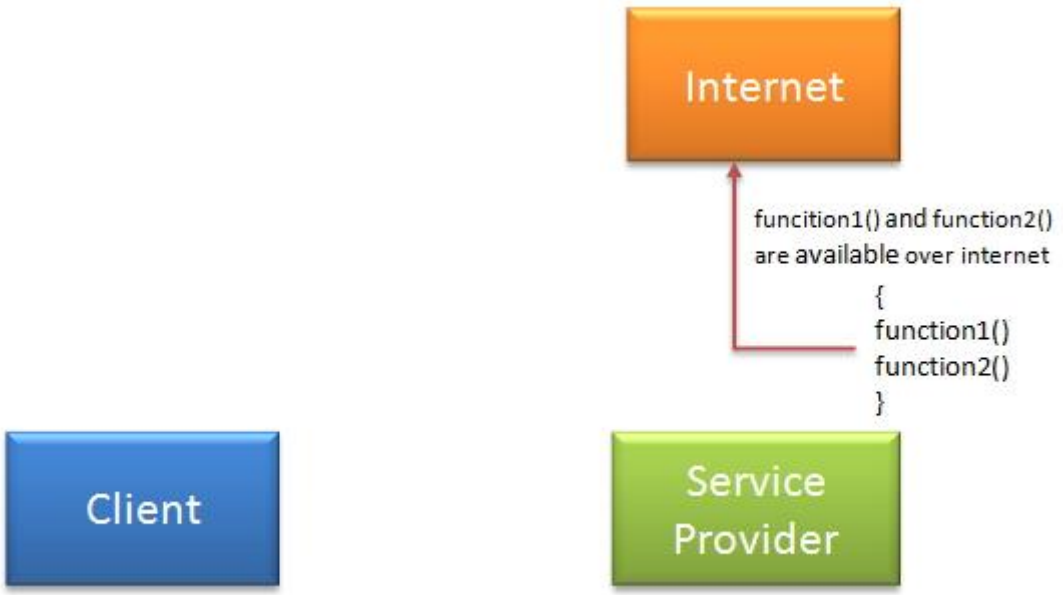
Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.

Cache: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
Named resources – the system is comprised of resources which are named using a URL.
Interconnected resource representations – the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.
Layered components – intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.
Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

SOAP web service::

Simple Object Access Protocol (SOAP) is a standard protocol specification for message exchange based on XML. Communication between the web service and client happens using XML messages.
A simple web service architecture have two components

- Client
- Service provider



So as in above diagram,how client will communicate to service provider.So in order to communicate client must know some information for e.g.

- Location of webservices server
- Functions available,signature and return types of function.
- Communication protocol
- Input output formats

Service provider will create a standard XML file which will have all above information.So If this file is given to client then client will be able to access web service. This XML file is called WSDL.

What is WSDL?

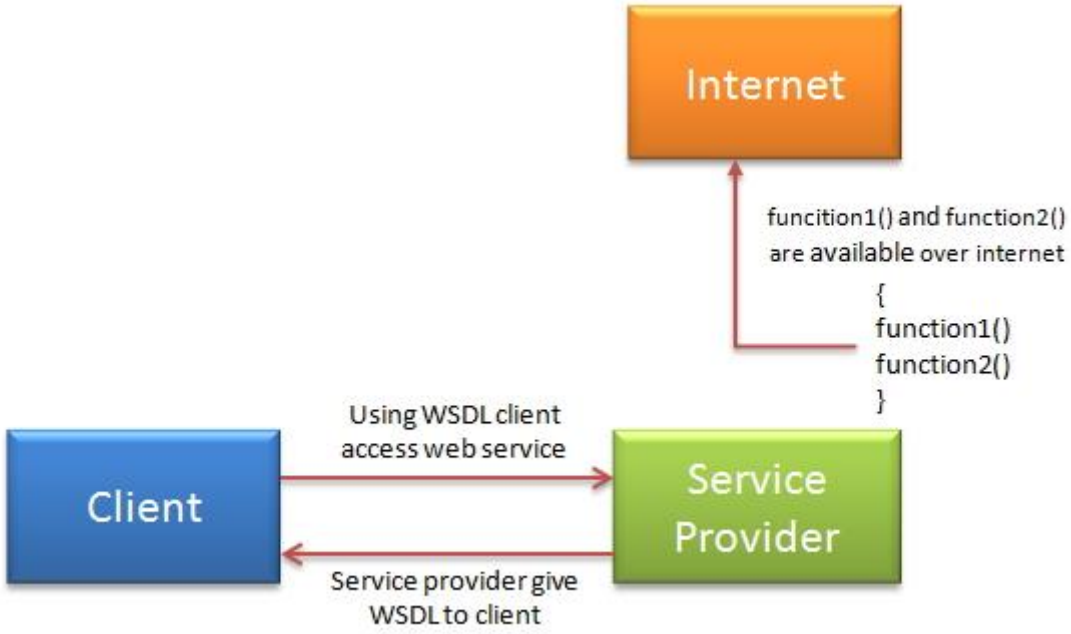
WSDL stands for Web Service Description Language. It is an XML file that describes
the technical details of how to implement a web service, more specifically the URI,
port, method names, arguments, and data types. Since WSDL is XML, it is both human-readable and machine-consumable, which aids in the ability to call and bind to
services dynamically.using this WSDL file we can understand things like,

- Port / Endpoint – URL of the web service
- Input message format
- Output message format
- Security protocol that needs to be followed
- Which protocol the web service uses

Ways to access web service:

There are two ways to access web service.

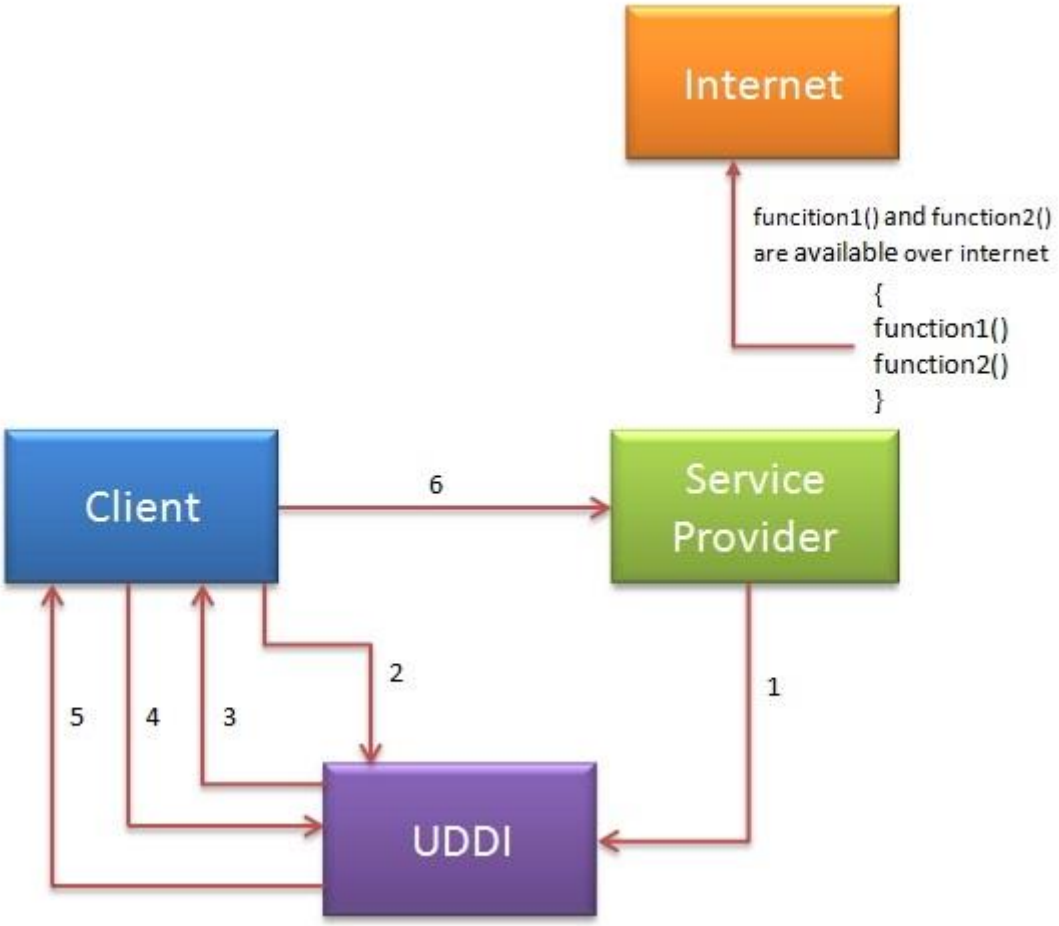
- **If Service provider knows client**:If service provider knows its client then it will provide its wsdl to client and client will be able to access web service.



- **Service provider register its WSDL to UDDI and client can access it from UDDI**:UDDI stands for Universal Description, Discovery and Integration.It is a directory service.

Web services can register with a UDDI and make themselves available through it for discovery.So following steps are involved.

1. Service provider registers with UDDI.
2. Client searches for service in UDDI.
3. UDDI returns all service providers offering that service.
4. Client chooses service provider
5. UDDI returns WSDL of chosen service provider.
6. Using WSDL of service provider,client accesses web service.



Difference between SOAP and REST web services in java::

We have already seen SOAP web services and RESTful web services in detail before. In this post, we are going to see differences between SOAP and REST web services.

SOAP vs REST web services

Parameter	SOAP	REST
Acronym	SOAP stands for simple object access protocol	REST stands for REpresentational State Transfer

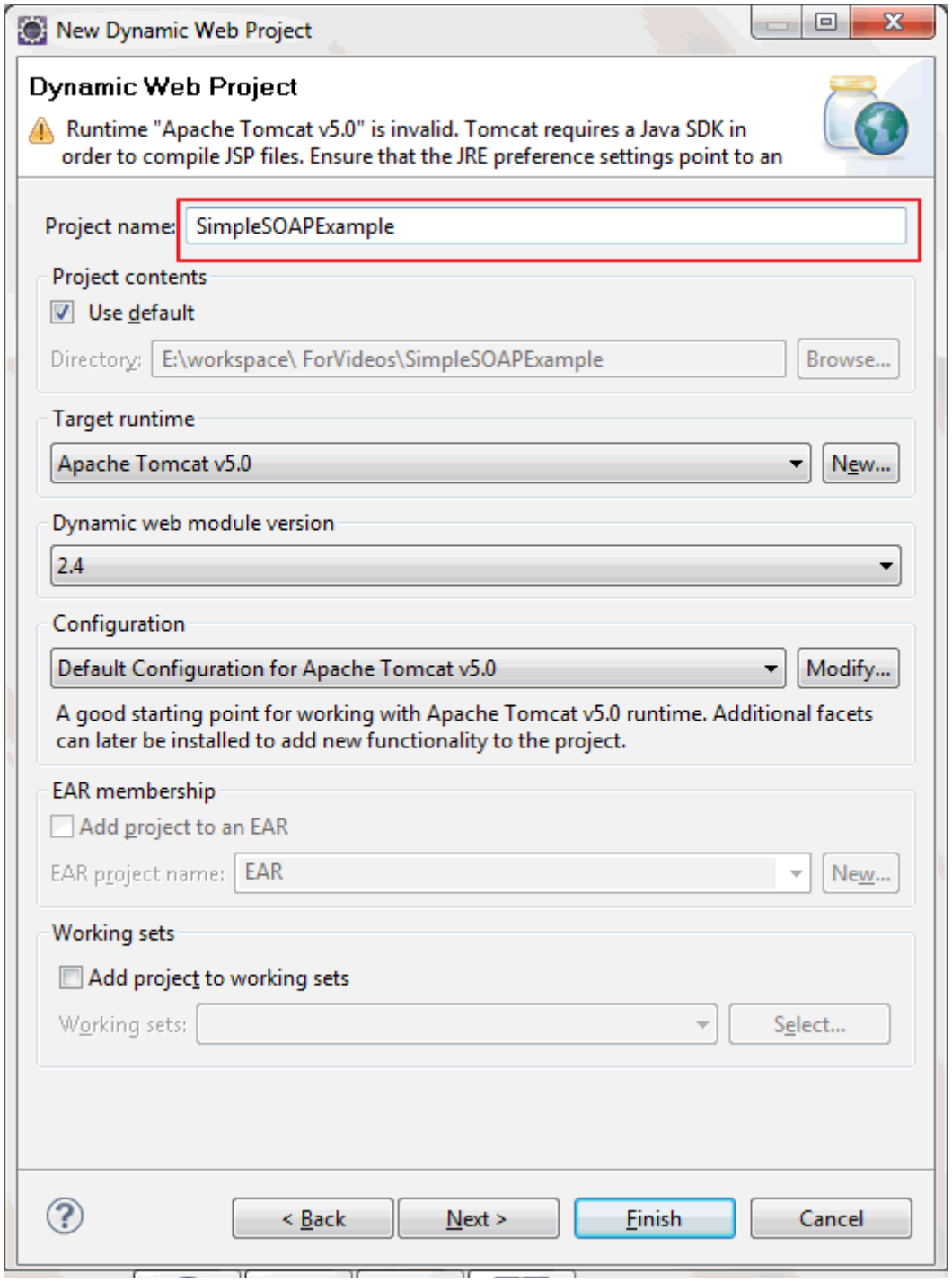
Protocol vs Architectural style	SOAP is a standard protocol to create web services	Rest is architectural style to create web services.
Contract	Client and Server are bind with WSDL contract	There is no contract between client and Server.
Format Support	SOAP supports only XML format	REST web services supports XML, json and plain text etc.
Maintainability	SOAP web services are hard to maintain as if we do any changes in WSDL , we need to create client stub again	REST web services are generally easy to maintain.
Service interfaces vs URI	SOAP uses Service interfaces to expose business logic	Rest uses URI to expose business logic
Security	SOAP has its own security : WS-security	Rest inherits its security from underlying transport layer.
Bandwidth	SOAP requires more bandwidth and resources as it uses XML messages to exchange information	REST requires less bandwith and resources. It can use JSON also.
Learning curve	SOAP web services are hard to learn as you need to understand WSDL , client stub	REST web services are easy to understand as you need to annotate plain java class with JAX-RS annotations to use various HTTP methods.

This is all about Difference between SOAP and REST web services in java.

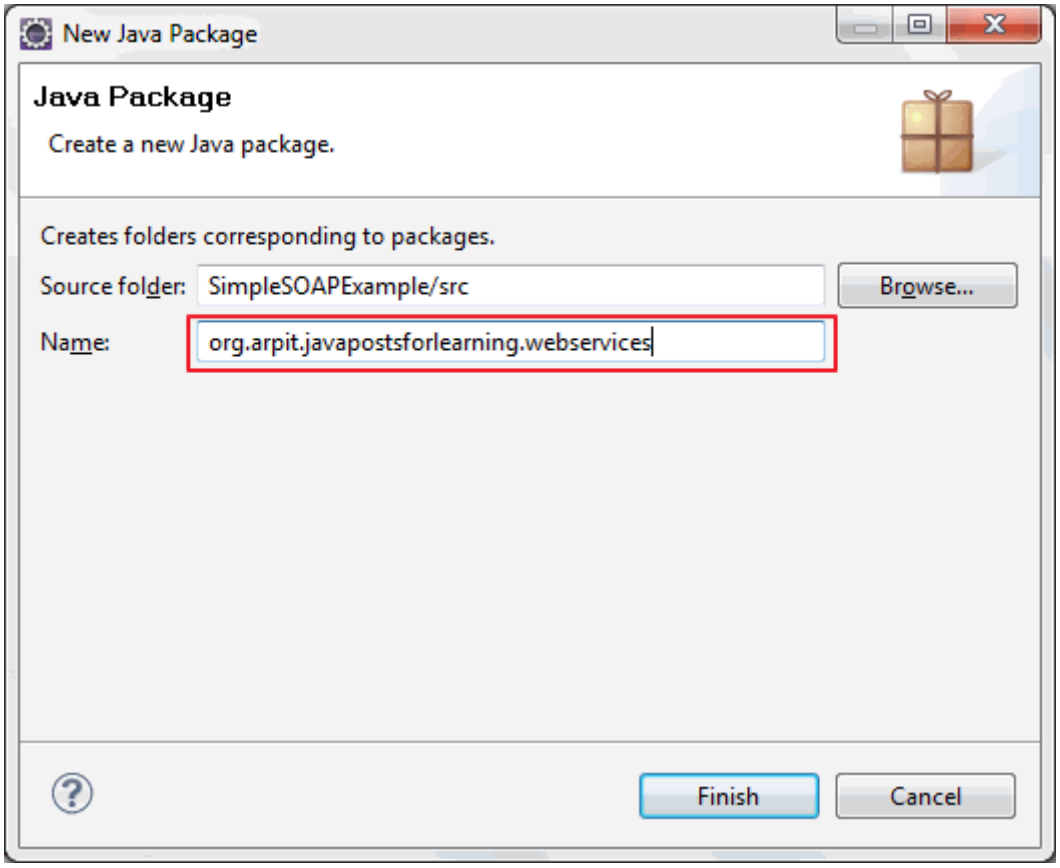
SOAP web service example in java using eclipse::

In this post,we will create hello world SOAP web service example in eclipse.Eclipse provides good API for creating web services.Eclipse will do all work for you-creating WSDL,stub,endpoints etc.
Steps for creating web services in eclipse:

1.Create new dynamic web project and name it “SimpleSOAPExample”.



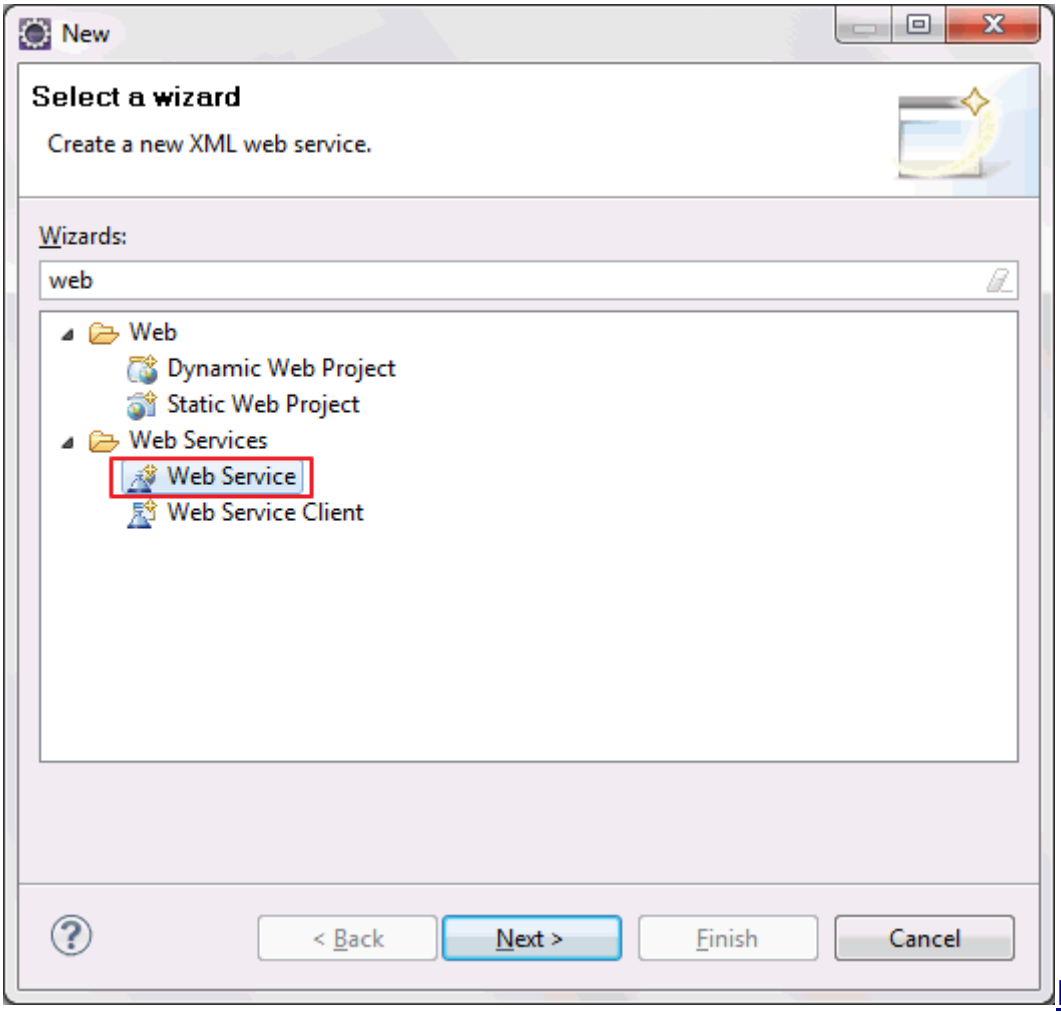
2.Create new package named “org.arpit.javapostsforlearning.webservices”



3.Create a simple java class named “HelloWorld.java”

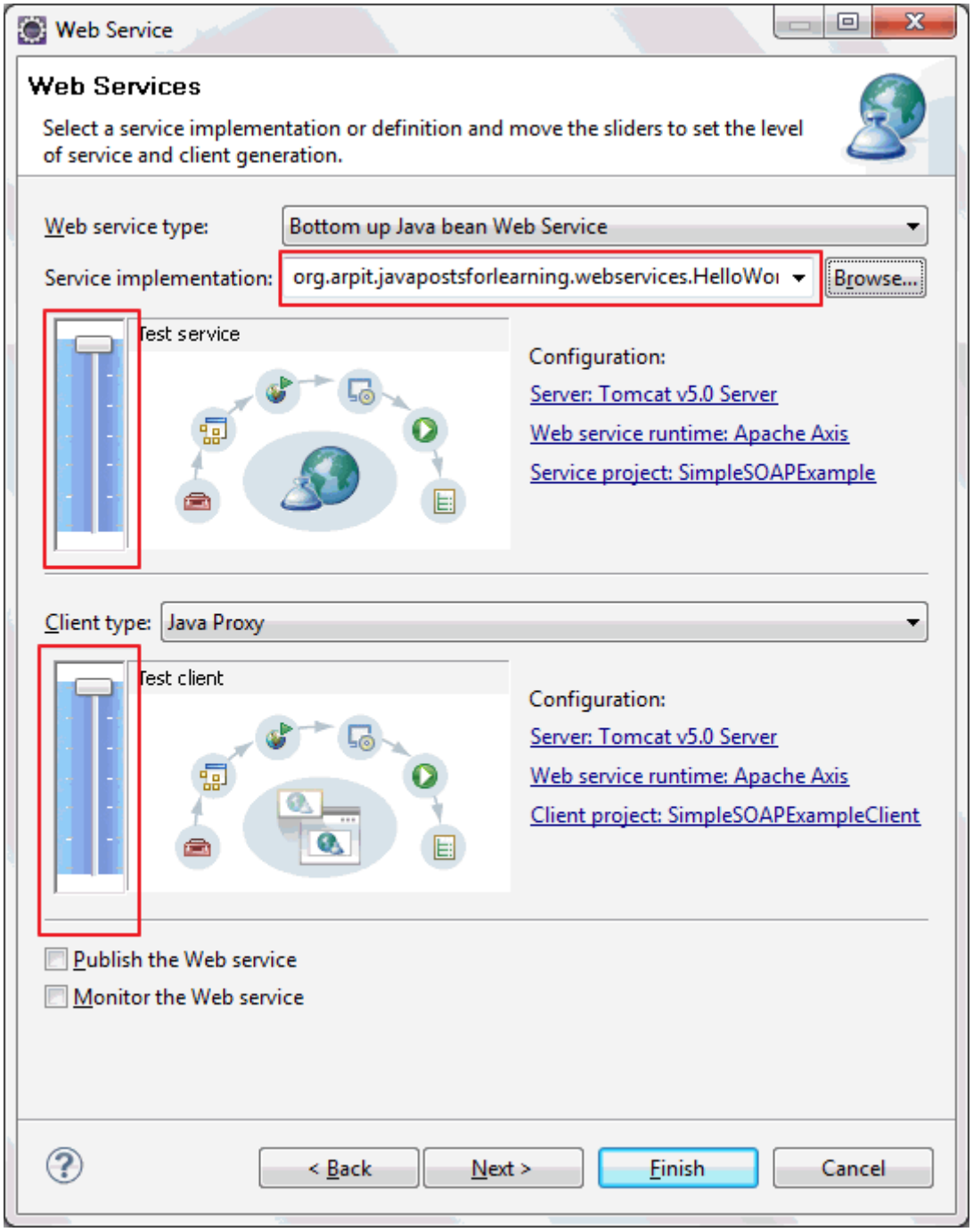
```
package org.arpit.javapostsforlearning.webservices;

public class HelloWorld {

 public String sayHelloWorld(String name)
 {
 return "Hello world from "+ name;
 }
}
```

4.Right click on project->new->web service

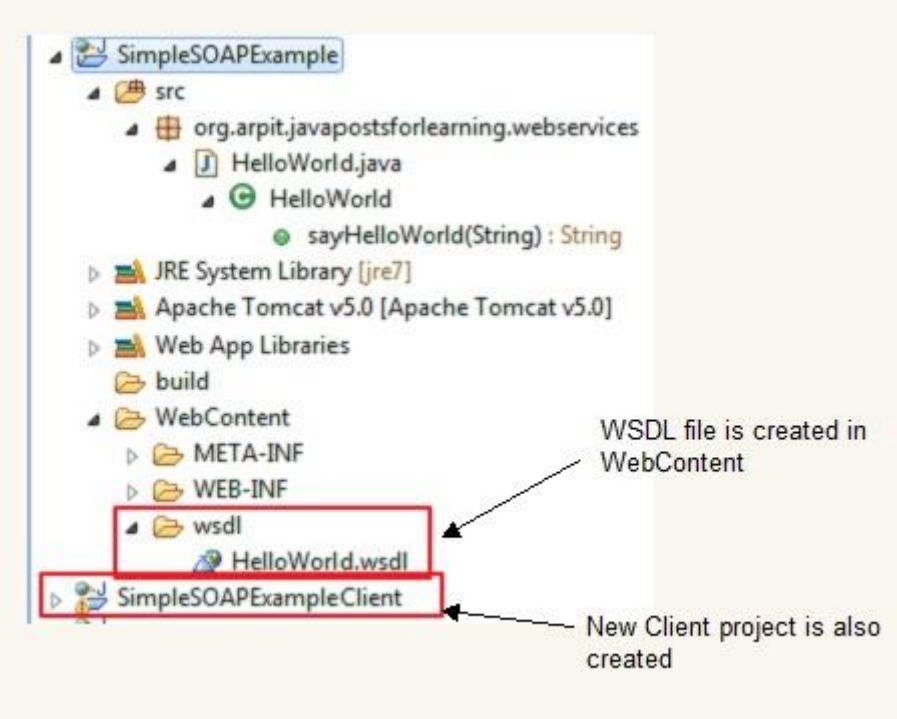


In

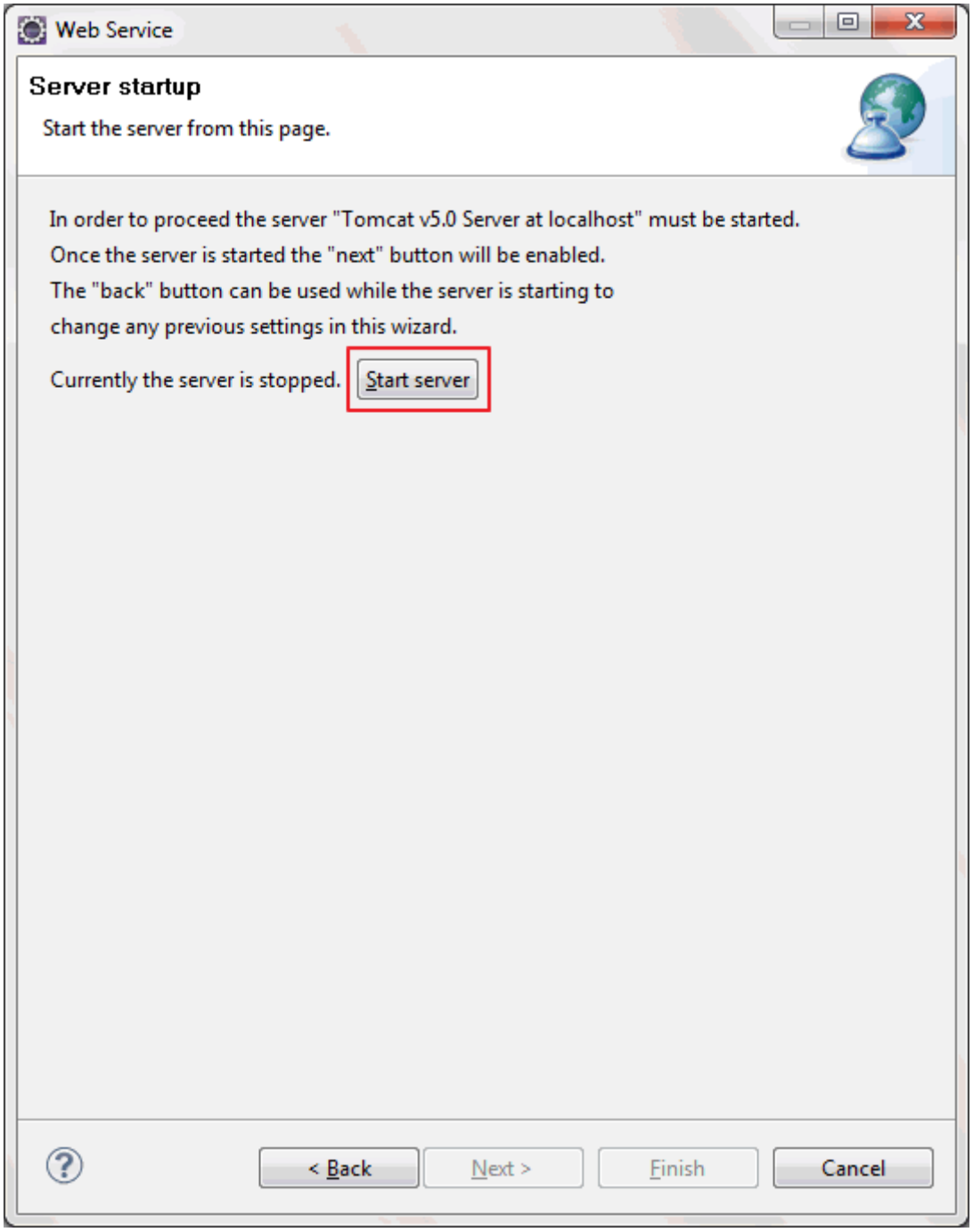
5.Click on next.



In service implementation text box,write fully qualified class name of above created class(HelloWorld.java) and move both above slider to maximum level (i.e. Test service and Test Client level) and click on finish.You are done!!A new project named “SimpleSOAPExampleClient” will be created in your work space.

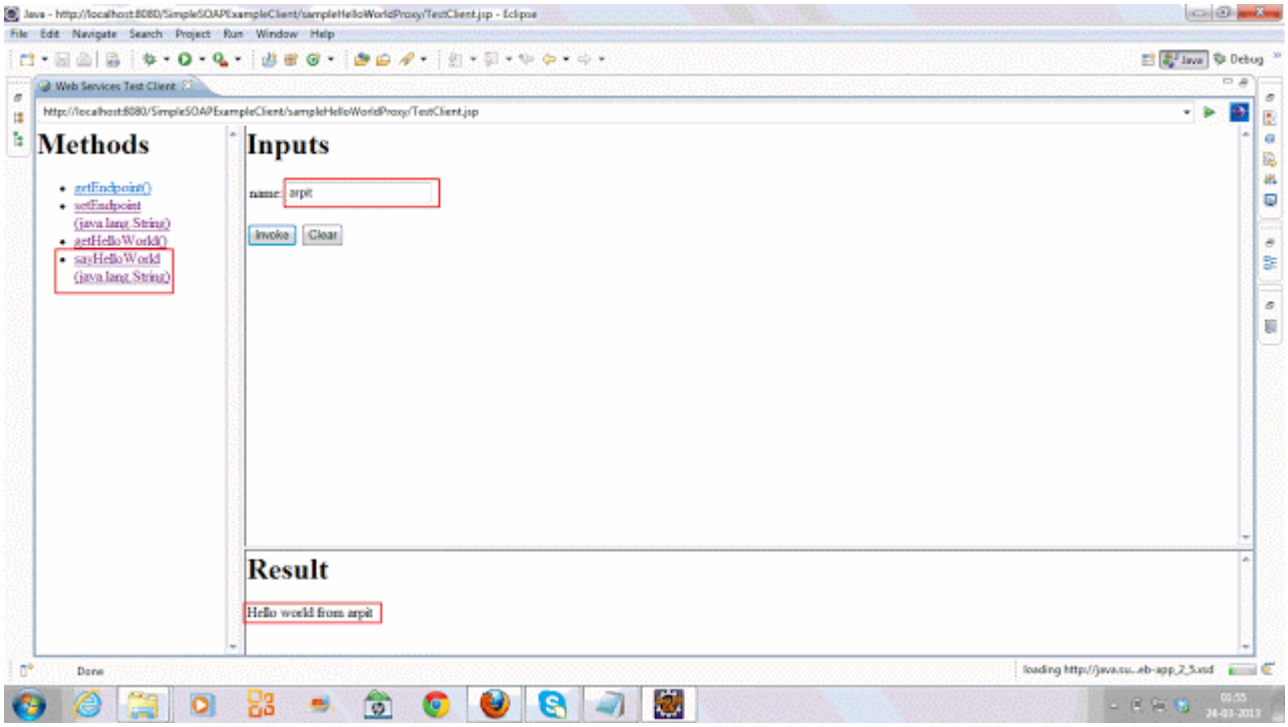

SimpleSOAPExample
src
org.arpit.javapostsforlearning.webservices
HelloWorld.java
HelloWorld
sayHelloWorld(String) : String
JRE System Library [jre7]
Apache Tomcat v5.0 [Apache Tomcat v5.0]
Web App Libraries
build
WebContent
META-INF
WEB-INF
wsdl
HelloWorld.wsdl
SimpleSOAPExampleClient
WSDL file is created in WebContent
New Client project is also created

6.



Click on start server.

7.After clicking start server,eclipse will open test web service API.With this test API,you can test your web service.



You are done!!.But to understand more about web services,you need to explore more.You can explore above created “SimpleSOAPExampleClient” and learn more about web services.

Source code :
Download

click to begin

6.2 MB .zip

Restful web services example in java::

Java API for RESTful Web Services (**JAX-RS**), is a set if APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.
Jersey is the reference implementation for this specification. Jersey contains basically a REST server and a REST client. The core client can communicate with the server using jersey lib.

Last updated : June 24th, 2018

2 COMMENTS

Restful web services example in java

Previous

Next

In this post, we will develop Restful web services example in java using jersey in eclipse

Web service Tutorial Content:

Java API for RESTful Web Services (**JAX-RS**), is a set if APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.

Jersey is the reference implementation for this specification. Jersey contains basically a REST server and a REST client. The core client can communicate with the server using jersey lib.

On the server side Jersey uses a servlet which scans predefined classes to identify RESTful resources. Via the web.xml configuration file for your web application.

The base URL of this servlet is:

```
1

2 <b>http://your_domain:port/display-name/url-pattern/path_from_rest_class </b>

3
```

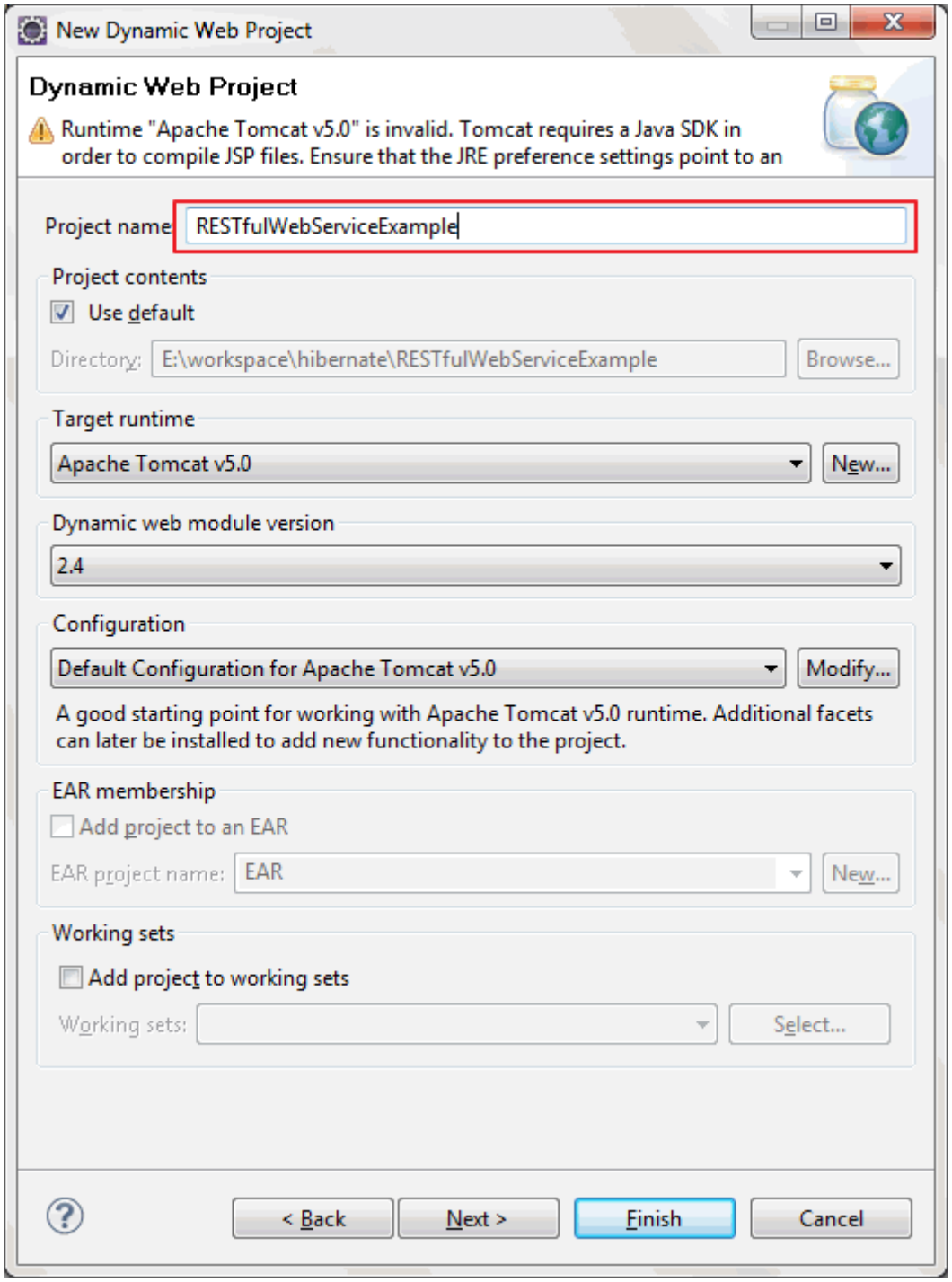
This servlet analyzes the incoming HTTP request and selects the correct class and method on request. This selection is based on annotations provided in the class and methods.

Let's see Restful web services example in java now.

Prerequisites:

- Java SE 6
- Download the zip of Jersey files from this location – https://jersey.java.net/download.html
- Eclipse IDE

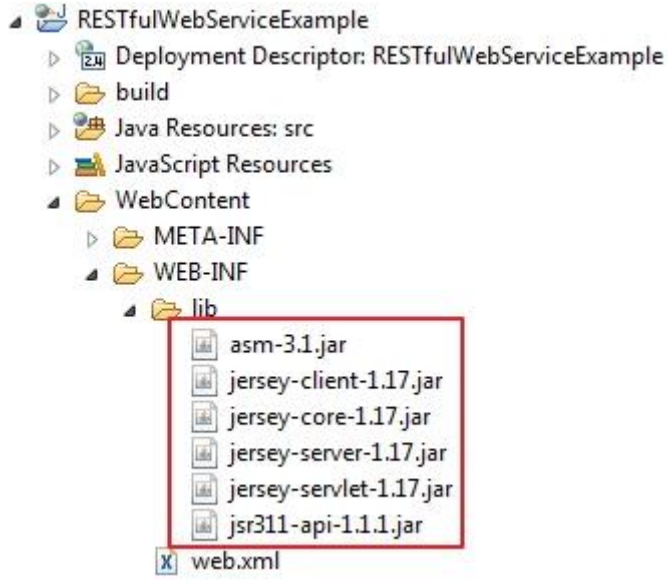
1) Open eclipse.
2) Create new dynamic web project named "RESTfulWebServiceExample"



3) Now go to location where you have download jersey and go to jersey-archive-1.17->lib
folder.you can have all jars but for now you can copy following jars

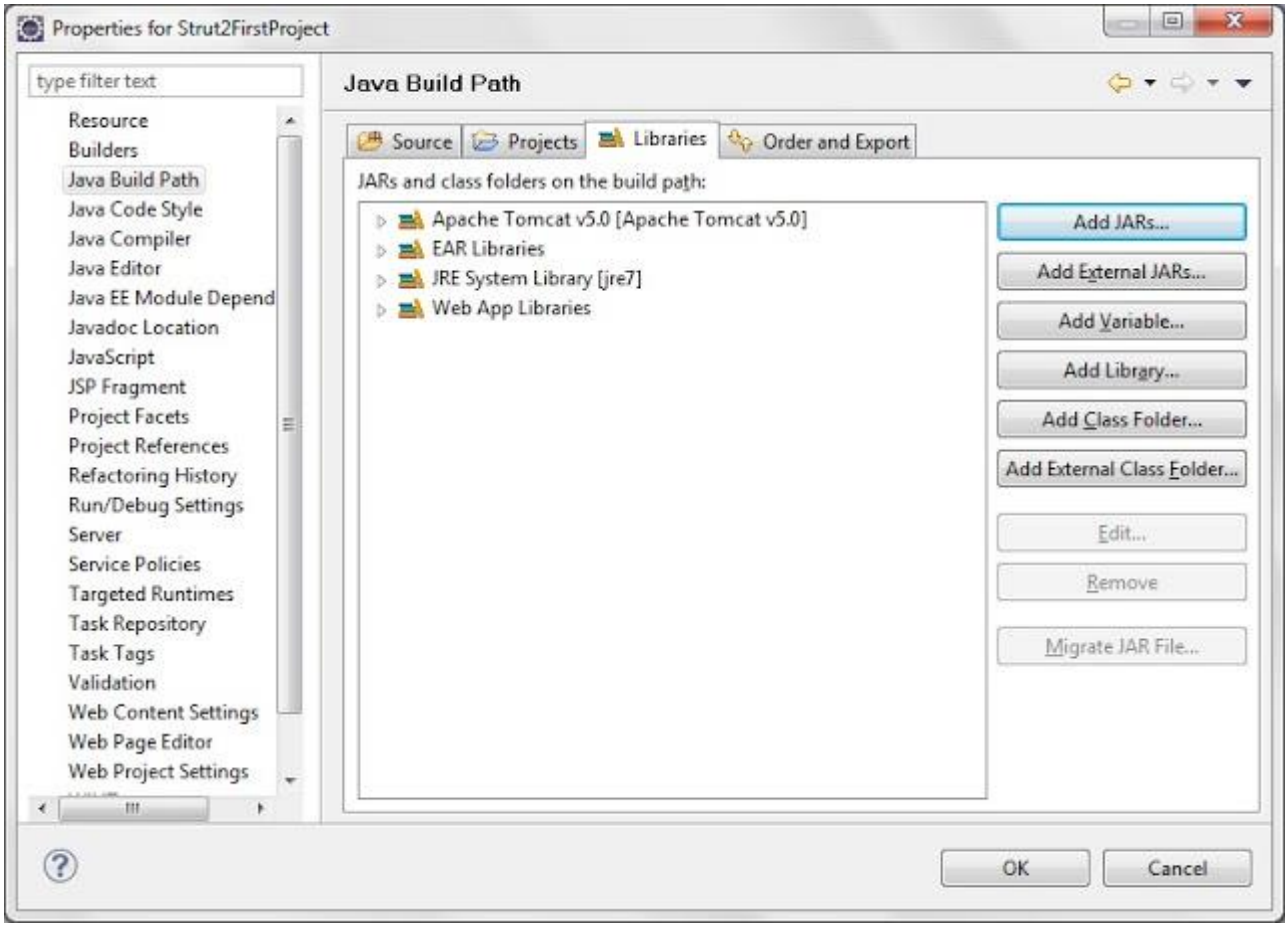
- asm-3.1
- jersey-client-1.17
- jersey-core-1.17
- jersey-server-1.17
- jersey-servlet-1.17
- jsr311-api-1.1.1

Paste all above copied jars to WebContent->WEB-INF->lib

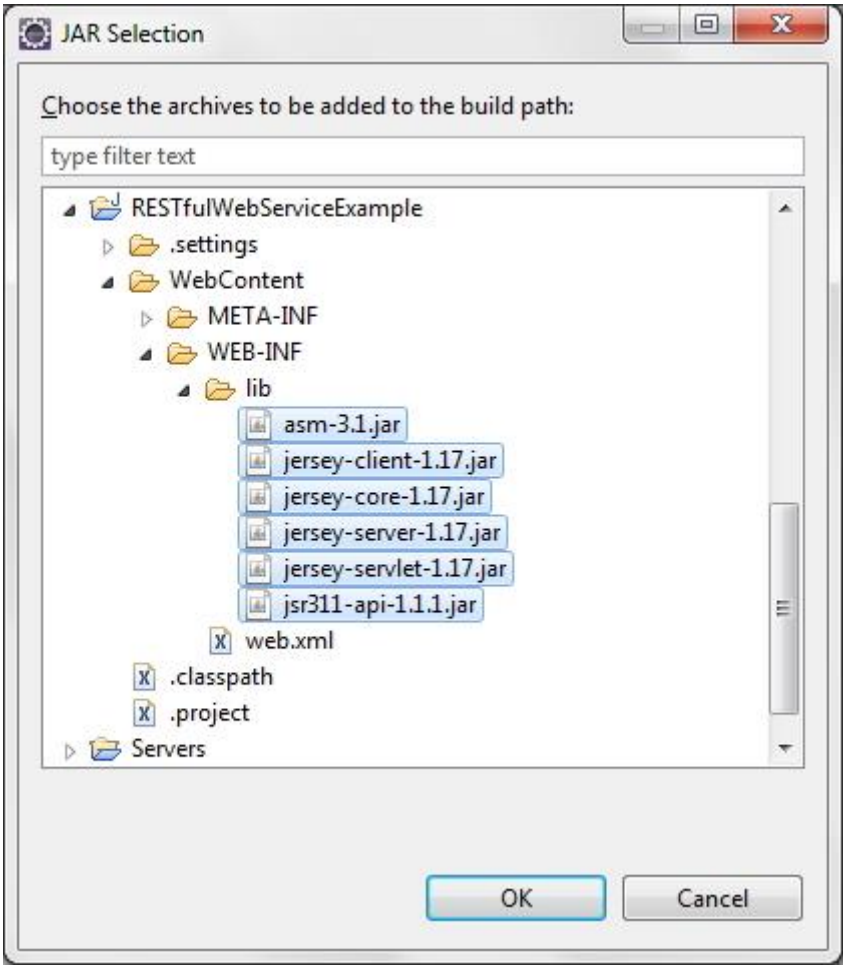


Add all these jars to eclipse build path.

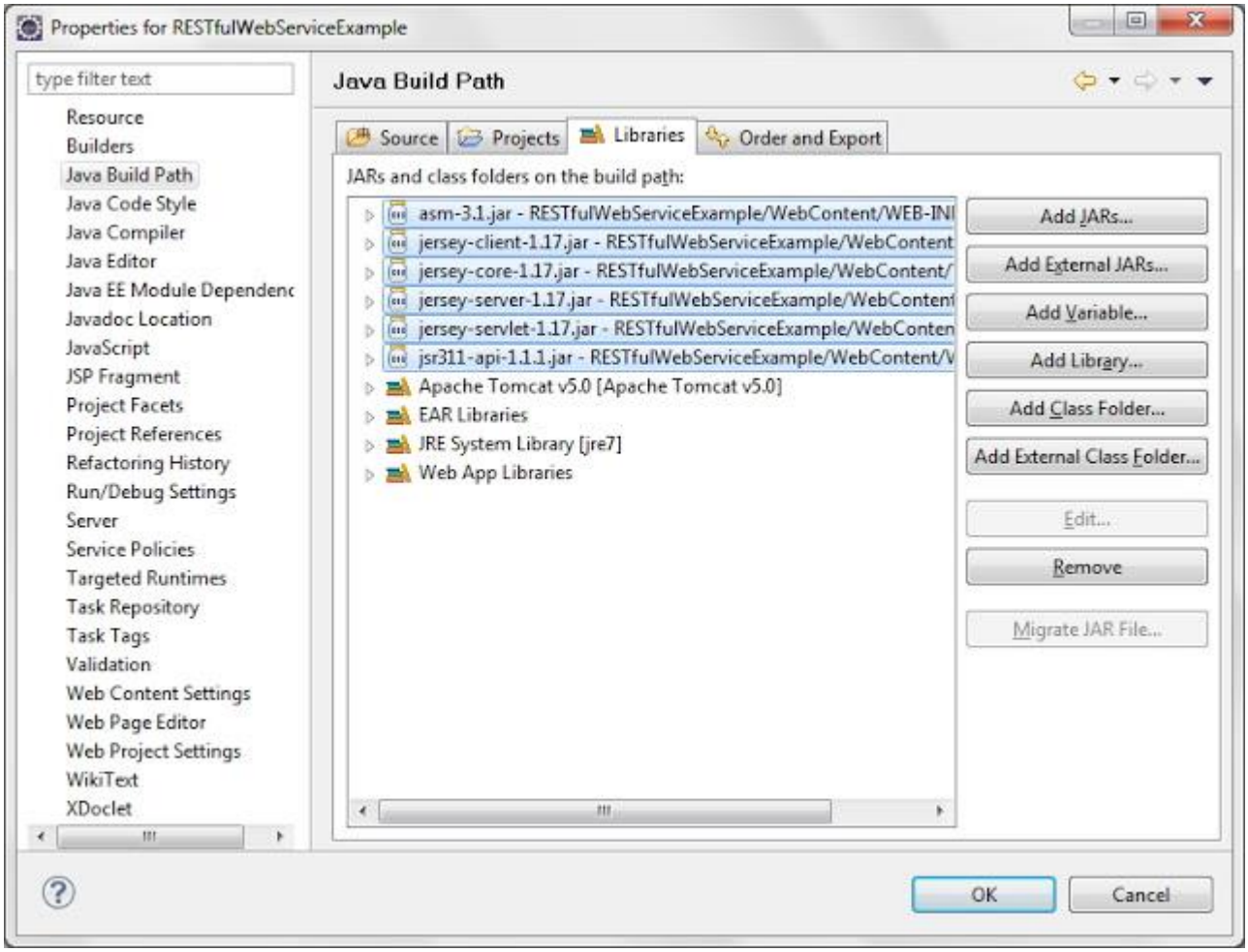
Right click on project(RESTfulWebServiceExample)->properties



Click on Java Build Path and then Add jars as shown in above diagram.

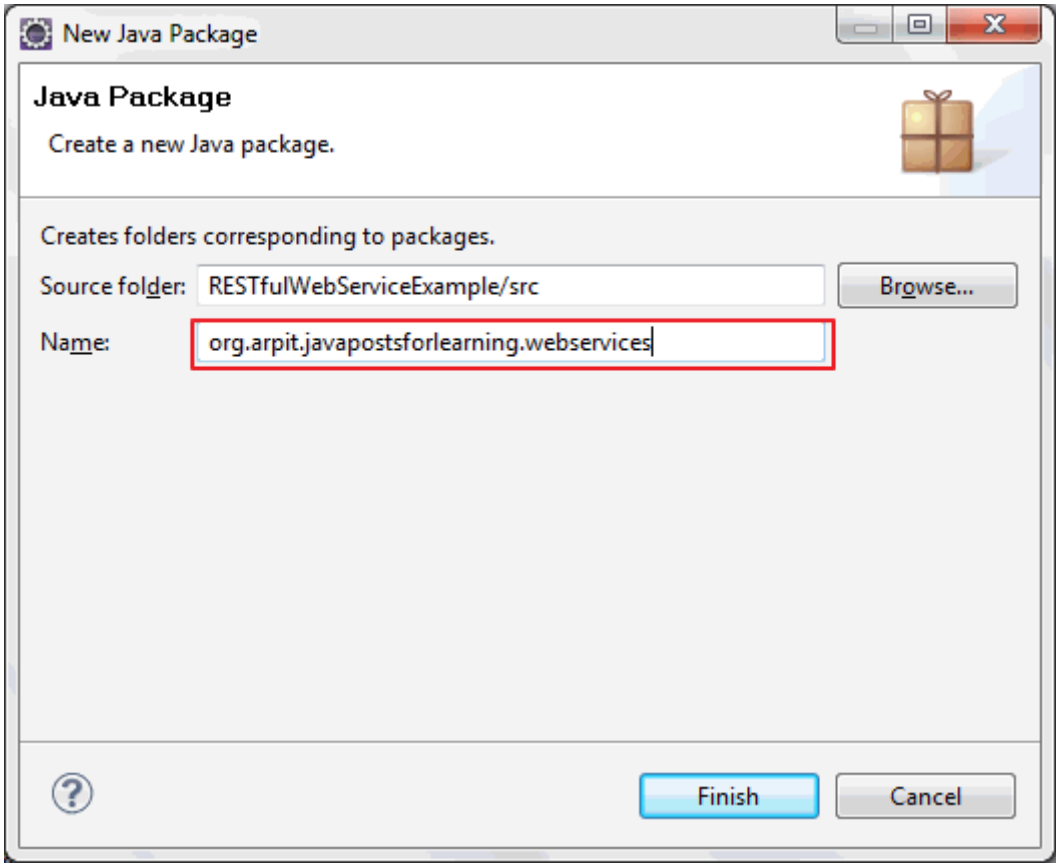


go to project->WebContent->WEB-INF->lib and select all jars then click on ok.



Click ok.Jersey jars added to class path.

4) Create new package named “org.arpit.javapostsforlearning.webservice”



5)Create **FeetToInchAndInchToFeetConversionService.java**

```
1
2  package org.arpit.javapostsforlearning.webservices;
3
4  /**
5  * @author Arpit Mandliya
6  */
7
8  import javax.ws.rs.GET;
9  import javax.ws.rs.Path;
10 import javax.ws.rs.PathParam;
11 import javax.ws.rs.Produces;
12 import javax.ws.rs.core.MediaType;
13
14 @Path("ConversionService")
15 public class FeetToInchAndInchToFeetConversionService {
16
```

```
17              @GET
18    @Path("/InchToFeet/{i}")
19     @Produces(MediaType.TEXT_XML)
20     public String convertInchToFeet(@PathParam("i") int i) {
21
22      int inch=i;
23      double feet = 0;
24      feet =(double) inch/12;
25
26      return "<InchToFeetService>"
27      + "<Inch>" + inch + "</Inch>"
28       + "<Feet>" + feet + "</Feet>"
29       + "</InchToFeetService>";
30     }
31
32     @Path("/FeetToInch/{f}")
33     @GET
34     @Produces(MediaType.TEXT_XML)
35     public String convertFeetToInch(@PathParam("f") int f) {
36              int inch=0;
37       int feet = f;
38       inch = 12*feet;
39
40       return "<FeetToInchService>"
41         + "<Feet>" + feet + "</Feet>"
42         + "<Inch>" + inch + "</Inch>"
43         + "</FeetToInchService>";
44     }
45
46
47 }
48
```

@Path(/your_path_at_class_level) : Sets the path to base URL + /your_path_at_class_level. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.

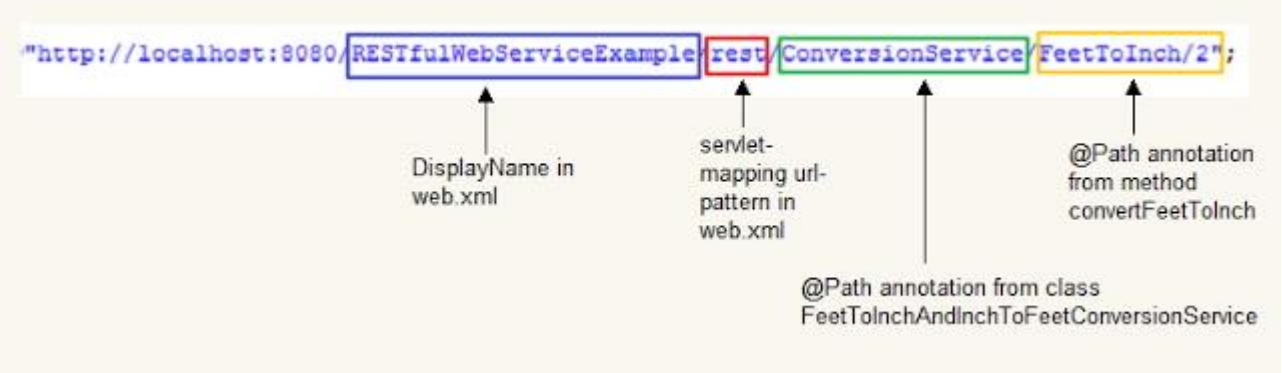
@Path(/your_path_at_method_level): Sets path to base URL + /your_path_at_class_level+ /your_path_at_method_level
@Produces(MediaType.TEXT_XML [, more-types]): @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/XML") is produced.
@PathParam: Used to inject values from the URL into a method parameter.This way you inject inch in convertFeetToInch method and convert that to feet.
6)Now you need to create **web.xml** and put it under **/RESTfulWebserviceExample/WebContent/WEB-INF/**

```
1  <!--?xml version="1.0" encoding="UTF-8"?-->
2  <?xml version="1.0" encoding="UTF-8"?>
3  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/ja
4  vaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.
   com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
5    <display-name>RESTfulWebServiceExample</display-name>
6    <servlet>
7      <servlet-name>Jersey REST Service</servlet-name>
8      <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
9      <init-param>
10       <param-name>com.sun.jersey.config.property.packages</param-name>
11       <param-value>org.arpit.javapostsforlearning.webservices</param-value>
12     </init-param>
13     <load-on-startup>1</load-on-startup>
14   </servlet>
15   <servlet-mapping>
16     <servlet-name>Jersey REST Service</servlet-name>
17     <url-pattern>/rest/*</url-pattern>
18   </servlet-mapping>
19 </web-app>
20
```

In above ,put your web service package.

8) **Run project:**right click on project->run as ->run on server

9) Test your REST service under: "http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/FeetToInch/2".



You will get output as :

If You see web service information page then you are done.

Creating a Restful Web Service Client:

Create ConversionServiceClient.java under org.arpit.javapostsforlearning.websevices.client

```
1
2  package org.arpit.javapostsforlearning.webservices.client;
3
4  import javax.ws.rs.core.MediaType;
5  import com.sun.jersey.api.client.Client;
6  import com.sun.jersey.api.client.ClientResponse;
7  import com.sun.jersey.api.client.WebResource;
8  import com.sun.jersey.api.client.config.ClientConfig;
9  import com.sun.jersey.api.client.config.DefaultClientConfig;
10
11 public class ConversionServiceClient {
12              static final String REST_URI = "http://localhost:8080/RESTfulWebServiceExample";
13              static final String INCH_TO_FEET = "/ConversionService/InchToFeet/";
```

```
14          static final String FEET_TO_INCH = "/ConversionService/FeetToInch/";
15
16          public static void main(String[] args) {
17
18                  int inch=12;
19                  int feet=2;
20
21                  ClientConfig config = new DefaultClientConfig();
22                  Client client = Client.create(config);
23                  WebResource service = client.resource(REST_URI);
24
25                  WebResource addService = service.path("rest").path(INCH_TO_FEET+inch);
26                  System.out.println("INCH_TO_FEET Response: " + getResponse(addService));
27                  System.out.println("INCH_TO_FEET Output as XML: " + getOutputAsXML(addService));
28                  System.out.println("-----------------------------------------------------");
29
30                  WebResource subService = service.path("rest").path(FEET_TO_INCH+feet);
31                  System.out.println("FEET_TO_INCH Response: " + getResponse(subService));
32                  System.out.println("FEET_TO_INCH Output as XML: " + getOutputAsXML(subService));
33                  System.out.println("-----------------------------------------------------");
34
35          }
36
37          private static String getResponse(WebResource service) {
38                  return service.accept(MediaType.TEXT_XML).get(ClientResponse.class).toString();
39          }
40
41          private static String getOutputAsXML(WebResource service) {
42                  return service.accept(MediaType.TEXT_XML).get(String.class);
43          }
44 }
45
```

Run above program
Output:

```
INCH_TO_FEET Response: GET http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/InchToFeet/12 returned a response status of 200 OK

INCH_TO_FEET Output as XML: 121.0

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

FEET_TO_INCH Response: GET http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/FeetToInch/2 returned a response status of 200 OK

FEET_TO_INCH Output as XML: 224

---------------------------------------------------
```

Source code :

Download

click to begin

1.2 MB .zip

In previous post, we have created a very simple Restful web services(JAXRS) using jersey which returns xml. In this post, we will see Restful web services(JAXRS) using jersey which will return json as response.
Here are steps to create a simple Restful web services(JAXRS) using jersey which will return json.
1) Create a dynamic web project using maven in eclipse named "JAXRSJsonExample"
2) We need to add jersey jars utility in the classpath.

```
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-servlet</artifactId>
  <version>${jersey.version}</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-json</artifactId>
  <version>${jersey.version}</version>
</dependency>
```

Jersey internally uses Jackson for Json Handling, so it will be used to marshal pojo objects to JSON.

Now create pom.xml as follows:
pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>com.arpit.java2blog</groupId>
6  <artifactId>JAXRSJsonExample</artifactId>
7  <packaging>war</packaging>
8  <version>0.0.1-SNAPSHOT</version>
9  <name>JAXRSJsonExample Maven Webapp</name>
10 <url>http://maven.apache.org</url>
11  <properties>
12      <jersey.version>1.18.3</jersey.version>
13  </properties>
14 <dependencies>
15  <dependency>
16  <groupId>junit</groupId>
17  <artifactId>junit</artifactId>
18  <version>3.8.1</version>
19  <scope>test</scope>
20 </dependency>
21
22 <dependency>
23  <groupId>com.sun.jersey</groupId>
24  <artifactId>jersey-servlet</artifactId>
25  <version>${jersey.version}</version>
26 </dependency>
27 <dependency>
28  <groupId>com.sun.jersey</groupId>
29  <artifactId>jersey-json</artifactId>
30  <version>${jersey.version}</version>
31 </dependency>
32
33 <dependency>
34  <groupId>commons-logging</groupId>
35  <artifactId>commons-logging</artifactId>
36  <version>1.2</version>
37 </dependency>
```

```
</dependencies>
<build>
 <finalName>JAXRSJsonExample</finalName>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>3.3</version>
   <configuration>
    <source>1.7</source>
    <target>1.7</target>
   </configuration>
  </plugin>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-war-plugin</artifactId>
   <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
   </configuration>


  </plugin>
 </plugins>
</build>

</project>

```

Application configuration:

3) create web.xml as below:

```

<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
   version="3.0">
```

```
 7   <display-name>Archetype Created Web Application</display-name>
 8   <servlet>
 9       <servlet-name>jersey-serlvet</servlet-name>
10       <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
11       <init-param>
12         <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
13         <param-value>true</param-value>
14       </init-param>
15       <load-on-startup>1</load-on-startup>
16   </servlet>
17
18   <servlet-mapping>
19       <servlet-name>jersey-serlvet</servlet-name>
20       <url-pattern>/rest/*</url-pattern>
21   </servlet-mapping>
22 </web-app>
```

Create bean class

4) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```
 1
 2  package org.arpit.java2blog.bean;
 3
 4  public class Country{
 5
 6   int id;
 7   String countryName;
 8
 9   public Country(int i, String countryName) {
10   super();
11   this.id = i;
12   this.countryName = countryName;
13   }
14   public int getId() {
15   return id;
16   }
```

```
17  public void setId(int id) {
18  this.id = id;
19  }
20  public String getCountryName() {
21  return countryName;
22  }
23  public void setCountryName(String countryName) {
24  this.countryName = countryName;
25  }
26
27 }
28
```

Create CountryRestService

5) Create a controller named “CountryRestService.java”

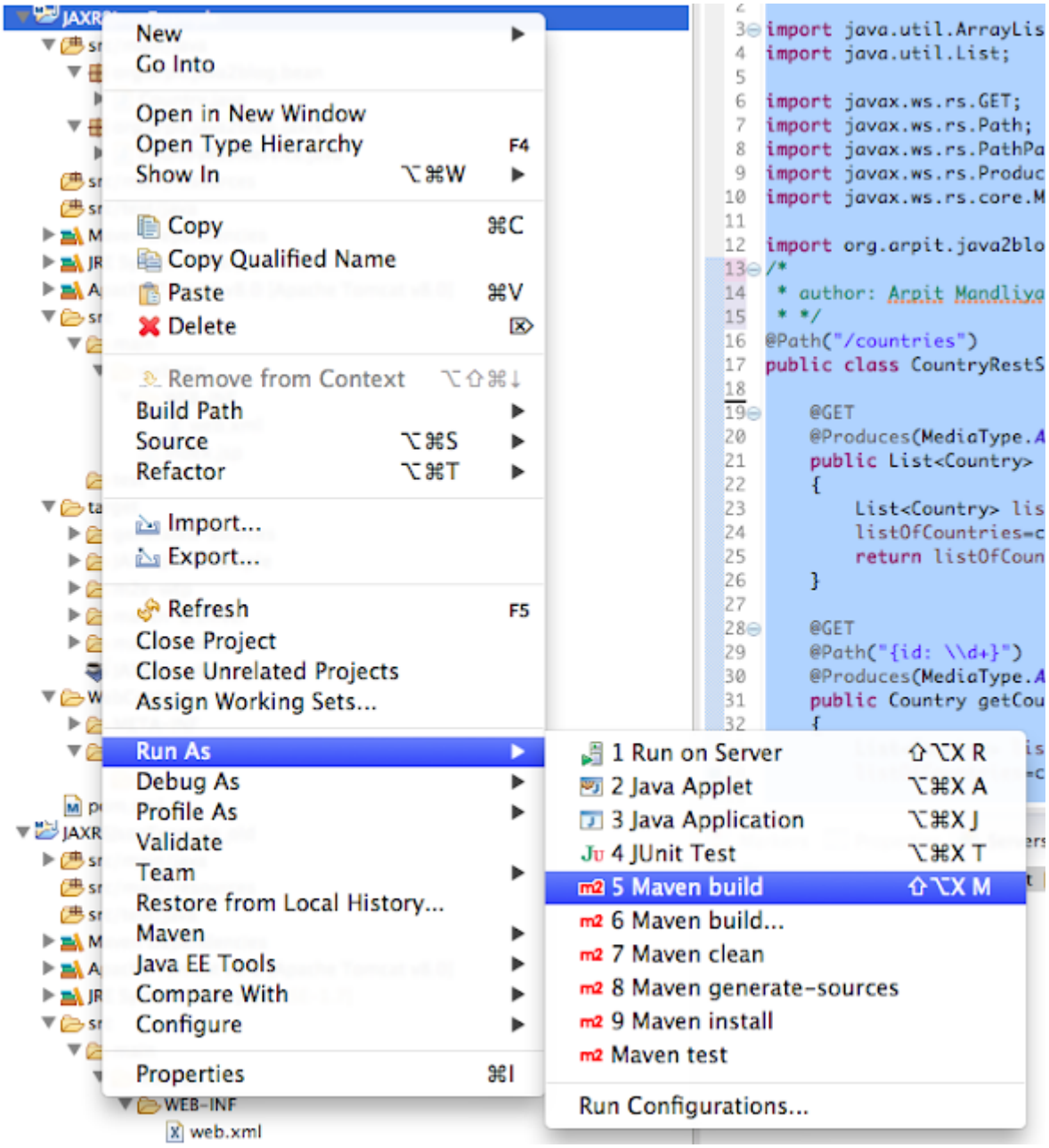
```
 1
 2  package org.arpit.java2blog.jaxrs;
 3
 4  import java.util.ArrayList;
 5  import java.util.List;
 6
 7  import javax.ws.rs.GET;
 8  import javax.ws.rs.Path;
 9  import javax.ws.rs.PathParam;
10 import javax.ws.rs.Produces;
11 import javax.ws.rs.core.MediaType;
12
13 import org.arpit.java2blog.bean.Country;
14 /*
15  * author: Arpit Mandliya
16  * */
17 @Path("/countries")
18 public class CountryRestService {
19
20    @GET
21    @Produces(MediaType.APPLICATION_JSON)
```

```
22  public List getCountries()
23  {
24   List listOfCountries = new ArrayList();
25   listOfCountries=createCountryList();
26   return listOfCountries;
27  }
28
29  @GET
30    @Path("{id: d+}")
31    @Produces(MediaType.APPLICATION_JSON)
32  public Country getCountryById(@PathParam("id") int id)
33  {
34   List listOfCountries = new ArrayList();
35   listOfCountries=createCountryList();
36
37   for (Country country: listOfCountries) {
38    if(country.getId()==id)
39     return country;
40   }
41
42   return null;
43  }
44
45 // Utiliy method to create country list.
46  public List createCountryList()
47  {
48   Country indiaCountry=new Country(1, "India");
49   Country chinaCountry=new Country(4, "China");
50   Country nepalCountry=new Country(3, "Nepal");
51   Country bhutanCountry=new Country(2, "Bhutan");
52
53   List listOfCountries = new ArrayList();
54   listOfCountries.add(indiaCountry);
55   listOfCountries.add(chinaCountry);
56   listOfCountries.add(nepalCountry);
```

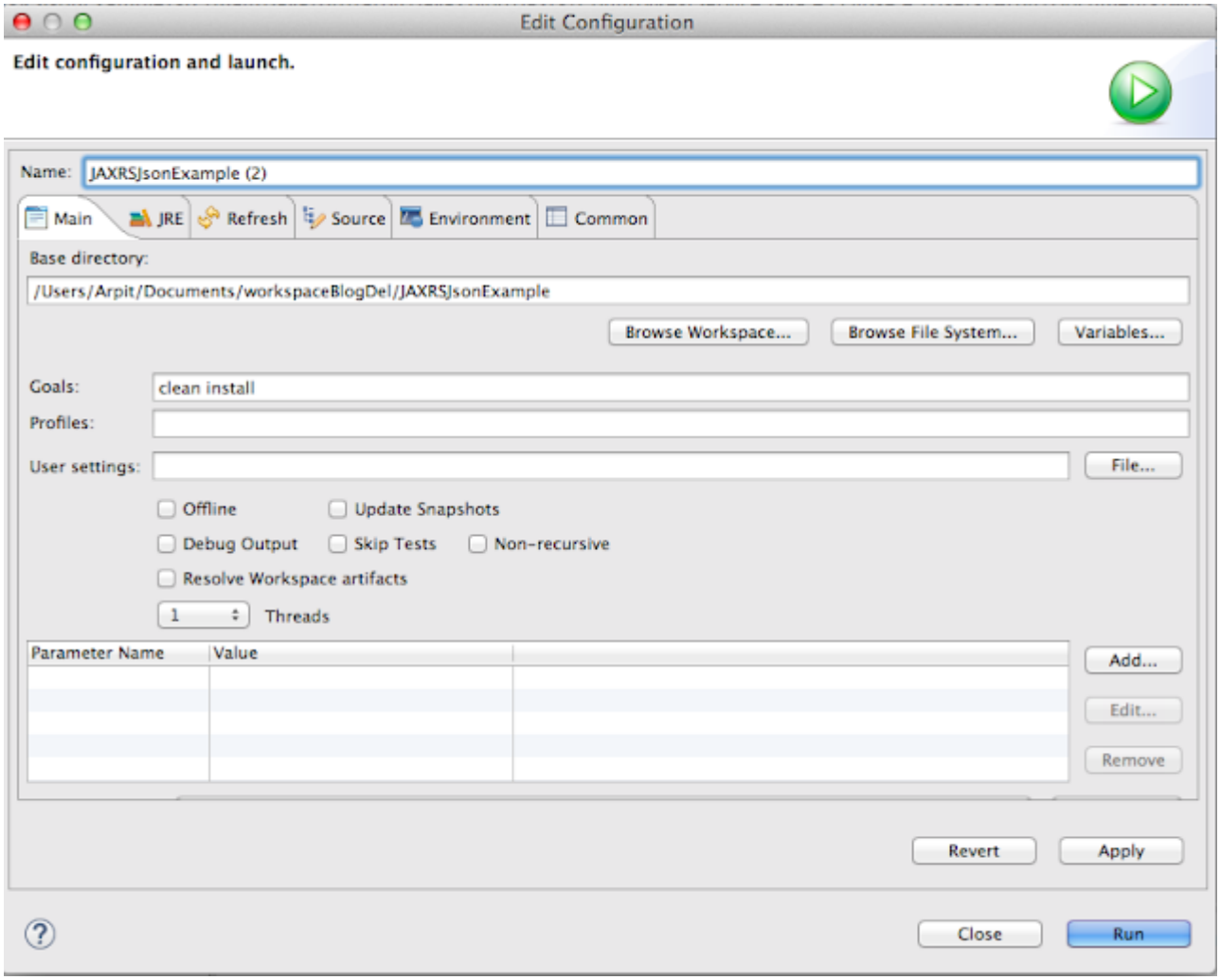
```
57   listOfCountries.add(bhutanCountry);
58   return listOfCountries;
59  }
60 }
61
```

@Path(/your_path_at_class_level) : Sets the path to base URL + /your_path_at_class_level. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.
@Path(/your_path_at_method_level): Sets path to base URL + /your_path_at_class_level+ /your_path_at_method_level
@Produces(MediaType.APPLICATION_JSON[, more-types]): @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/json") is produced.

6) It 's time to do maven build.
Right click on project -> Run as -> Maven build



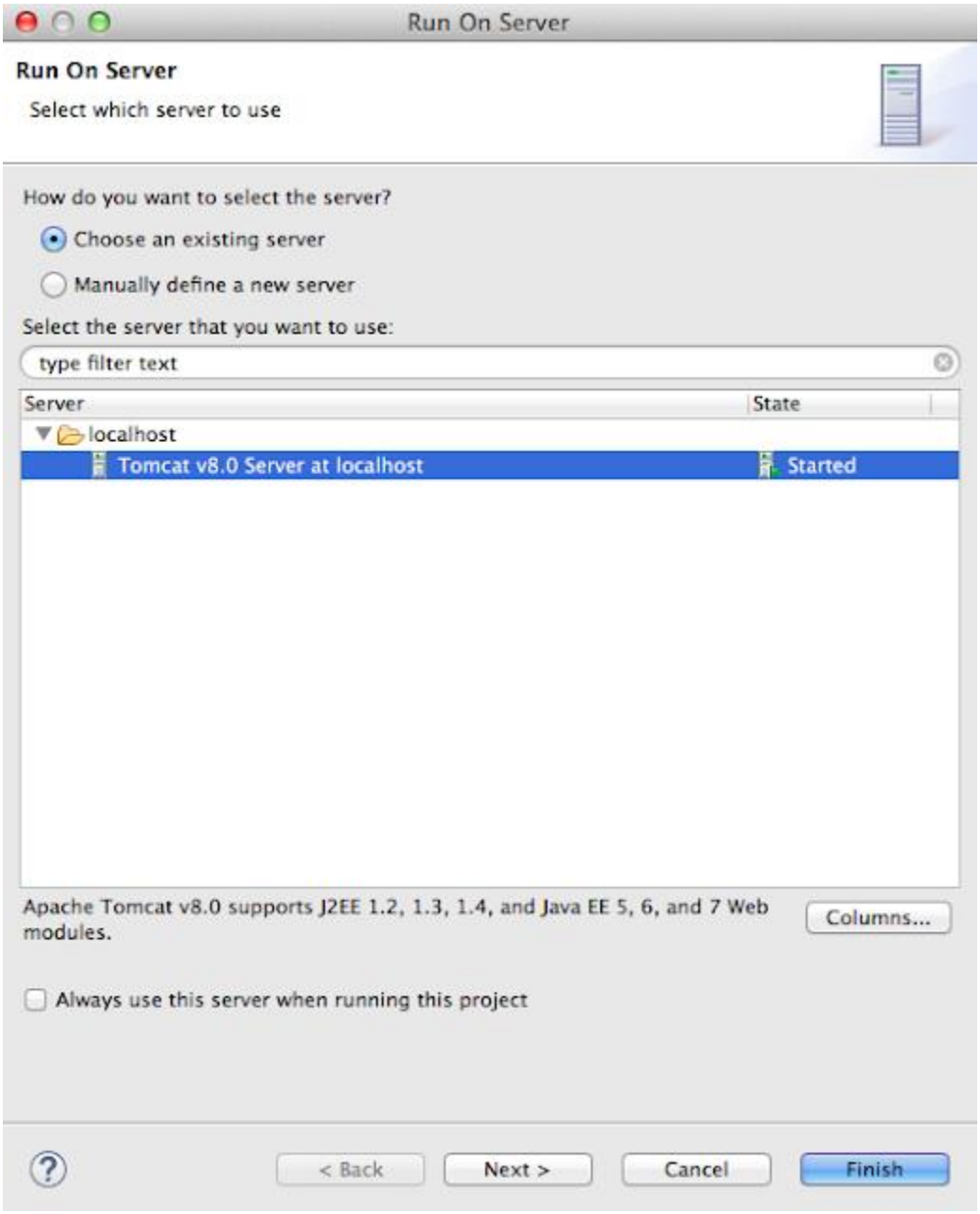
7) Provide goals as clean install (given below) and click on run



Run the application

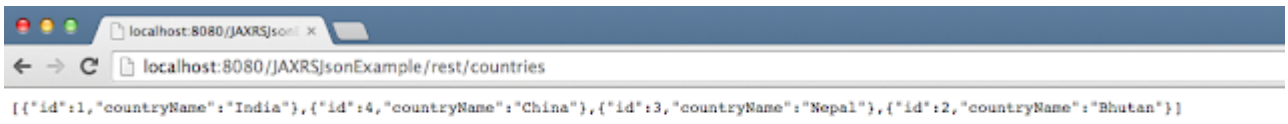
8) Right click on project -> run as -> run on server

Select apache tomcat and click on finish

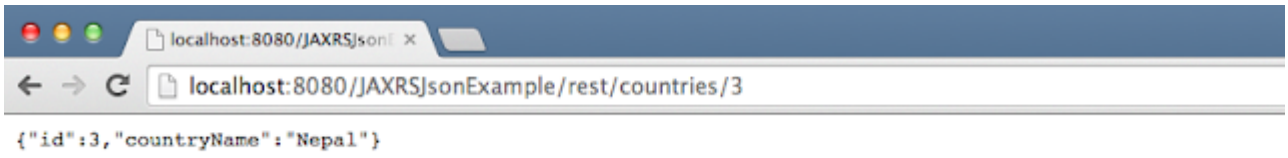


9) Test your REST service under: "http://localhost:8080/JAXRSJsonExample/rest/countries".

You will get following output:



10) Now pass country id as parameter to url. “http://localhost:8080/JAXRSJsonExample/rest/countries/3”.

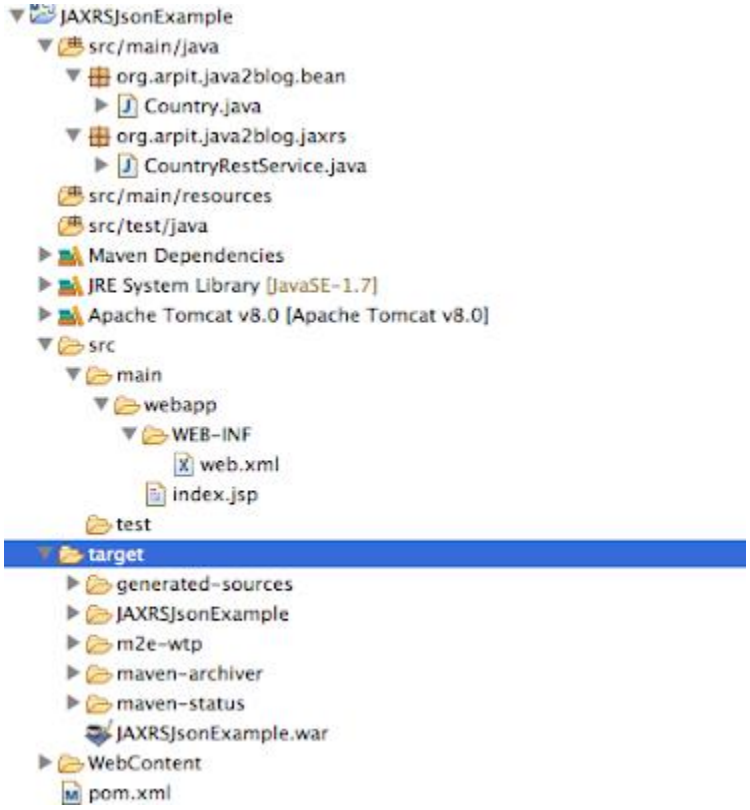


Download

click to begin

20KB .zip

Project structure:



We are done with Restful web services json example using jersey.

RESTful web services JAXRS CRUD example using jersey::

In previous post, we have already seen simple Restful web services(JAXWS) which returns json as response.In this post, we will extend same example and create Restful web services(JAXWS) using jersey which will provide CRUD(Create, read, update and delete) operation example.

We will use following annotations for CRUD operation.

Method	Description
Get	It is used to read resource
Post	It is used to create new resource. It is not idempotent method
Put	It is generally used to update resource idempotent method
Delete	It is used to delete resource

Idempotent means result of multiple successful request will not change state of resource after initial application
For example :
Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.

Post is not idempotent method because when you use post to create resource , it will keep creating resource for each new request, so result of multiple successful request will not be same.

Source code:

Download

click to begin

20KB .zip

Here are steps to create a simple Restful web services(JAXWS) using jersey which will provide CRUD opertion.
1) Create a dynamic web project using maven in eclipse named "JAXRSJsonCRUDExample"

Maven dependencies

2) We need to add jersey jars utility in the classpath.

```
1
2
3  <dependency>
4    <groupId>com.sun.jersey</groupId>
5    <artifactId>jersey-servlet</artifactId>
6    <version>${jersey.version}</version>
7  </dependency>
8  <dependency>
9    <groupId>com.sun.jersey</groupId>
10   <artifactId>jersey-json</artifactId>
11   <version>${jersey.version}</version>
12 </dependency>
13
```

Jersey internally uses Jackson for Json Handling, so it will be used to marshal pojo objects to JSON.

Now create pom.xml as follows:
pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
3  nce"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>com.arpit.java2blog</groupId>
7   <artifactId>JAXRSJsonExample</artifactId>
8   <packaging>war</packaging>
9   <version>0.0.1-SNAPSHOT</version>
10  <name>JAXRSJsonExample Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12   <properties>
13       <jersey.version>1.18.3</jersey.version>
14    </properties>
15  <dependencies>
16   <dependency>
17    <groupId>junit</groupId>
18    <artifactId>junit</artifactId>
```

```
19   <version>3.8.1</version>
20   <scope>test</scope>
21  </dependency>
22
23  <dependency>
24   <groupId>com.sun.jersey</groupId>
25   <artifactId>jersey-servlet</artifactId>
26   <version>${jersey.version}</version>
27  </dependency>
28  <dependency>
29   <groupId>com.sun.jersey</groupId>
30   <artifactId>jersey-json</artifactId>
31   <version>${jersey.version}</version>
32  </dependency>
33
34  <dependency>
35   <groupId>commons-logging</groupId>
36   <artifactId>commons-logging</artifactId>
37   <version>1.2</version>
38  </dependency>
39 </dependencies>
40 <build>
41  <finalName>JAXRSJsonExample</finalName>
42  <plugins>
43   <plugin>
44    <groupId>org.apache.maven.plugins</groupId>
45    <artifactId>maven-compiler-plugin</artifactId>
46    <version>3.3</version>
47    <configuration>
48     <source>1.7</source>
49     <target>1.7</target>
50    </configuration>
51   </plugin>
52   <plugin>
53    <groupId>org.apache.maven.plugins</groupId>
```

```
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
     <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
   </plugin>
  </plugins>
 </build>

</project>
```

Application configuration:

3) create web.xml as below:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns: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_3_0.xsd
"
 version="3.0">
 <display-name>Archetype Created Web Application</display-name>
 <servlet>
  <servlet-name>jersey-serlvet</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>org.arpit.java2blog.controller</param-value>
  </init-param>
  <init-param>
   <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
   <param-value>true</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
```

```
24   <servlet-name>jersey-serlvet</servlet-name>
25   <url-pattern>/rest/*</url-pattern>
26  </servlet-mapping>
27 </web-app>
```

Please change initParam "**com.sun.jersey.config.property.package**" property to provide correct controller package name if you are not using same package.

Create bean class

4) Create a bean name "Country.java" in org.arpit.java2blog.bean.

```
 1
 2  package org.arpit.java2blog.bean;
 3
 4  public class Country{
 5
 6   int id;
 7   String countryName;
 8   long population;
 9
10   public Country() {
11   super();
12   }
13   public Country(int i, String countryName,long population) {
14   super();
15   this.id = i;
16   this.countryName = countryName;
17   this.population=population;
18   }
19   public int getId() {
20   return id;
21   }
22   public void setId(int id) {
23   this.id = id;
24   }
25   public String getCountryName() {
```

```
26  return countryName;
27  }
28  public void setCountryName(String countryName) {
29  this.countryName = countryName;
30  }
31  public long getPopulation() {
32  return population;
33  }
34  public void setPopulation(long population) {
35  this.population = population;
36  }
37
38 }
39
```

Create Controller

5) Create a controller named “CountryController.java” in package **org.arpit.java2blog.controller**

```
 1
 2  package org.arpit.java2blog.controller;
 3
 4  import java.util.List;
 5
 6  import javax.ws.rs.DELETE;
 7  import javax.ws.rs.GET;
 8  import javax.ws.rs.POST;
 9  import javax.ws.rs.PUT;
10 import javax.ws.rs.Path;
11 import javax.ws.rs.PathParam;
12 import javax.ws.rs.Produces;
13 import javax.ws.rs.core.MediaType;
14
15 import org.arpit.java2blog.bean.Country;
16 import org.arpit.java2blog.service.CountryService;
17
18
```

```
19 @Path("/countries")
20 public class CountryController {
21
22  CountryService countryService=new CountryService();
23
24    @GET
25    @Produces(MediaType.APPLICATION_JSON)
26  public List getCountries()
27  {
28
29   List listOfCountries=countryService.getAllCountries();
30   return listOfCountries;
31  }
32
33    @GET
34    @Path("/{id}")
35    @Produces(MediaType.APPLICATION_JSON)
36  public Country getCountryById(@PathParam("id") int id)
37  {
38   return countryService.getCountry(id);
39  }
40
41    @POST
42    @Produces(MediaType.APPLICATION_JSON)
43  public Country addCountry(Country country)
44  {
45   return countryService.addCountry(country);
46  }
47
48    @PUT
49    @Produces(MediaType.APPLICATION_JSON)
50  public Country updateCountry(Country country)
51  {
52   return countryService.updateCountry(country);
53
```

```
54 }
55
56    @DELETE
57    @Path("/{id}")
58    @Produces(MediaType.APPLICATION_JSON)
59  public void deleteCountry(@PathParam("id") int id)
60  {
61    countryService.deleteCountry(id);
62
63  }
64
65 }
66
```

@Path(/your_path_at_class_level) : Sets the path to base URL + /your_path_at_class_level. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.
@Path(/your_path_at_method_level): Sets path to base URL + /your_path_at_class_level+ /your_path_at_method_level
@Produces(MediaType.APPLICATION_JSON[, more-types]): @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/json") is produced.

Create Service class

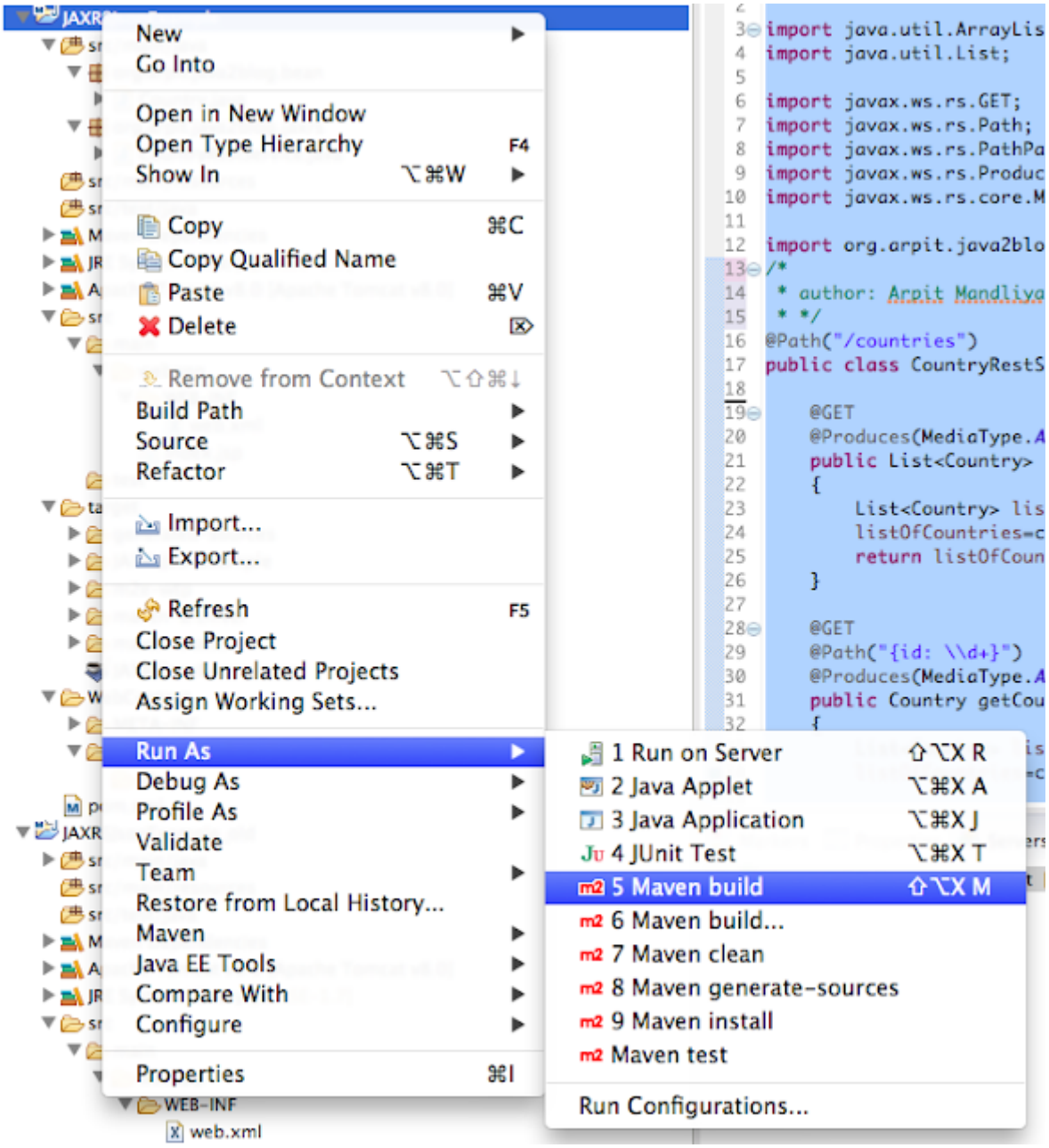
6) Create a class CountryService.java in package **org.arpit.java2blog.service**
It is just a helper class which should be replaced by database implementation. It is not very well written class, it is just used for demonstration.

```
1
2  package org.arpit.java2blog.service;
3
4  import java.util.ArrayList;
5  import java.util.HashMap;
6  import java.util.List;
7
8  import org.arpit.java2blog.bean.Country;
9
10 /*
```

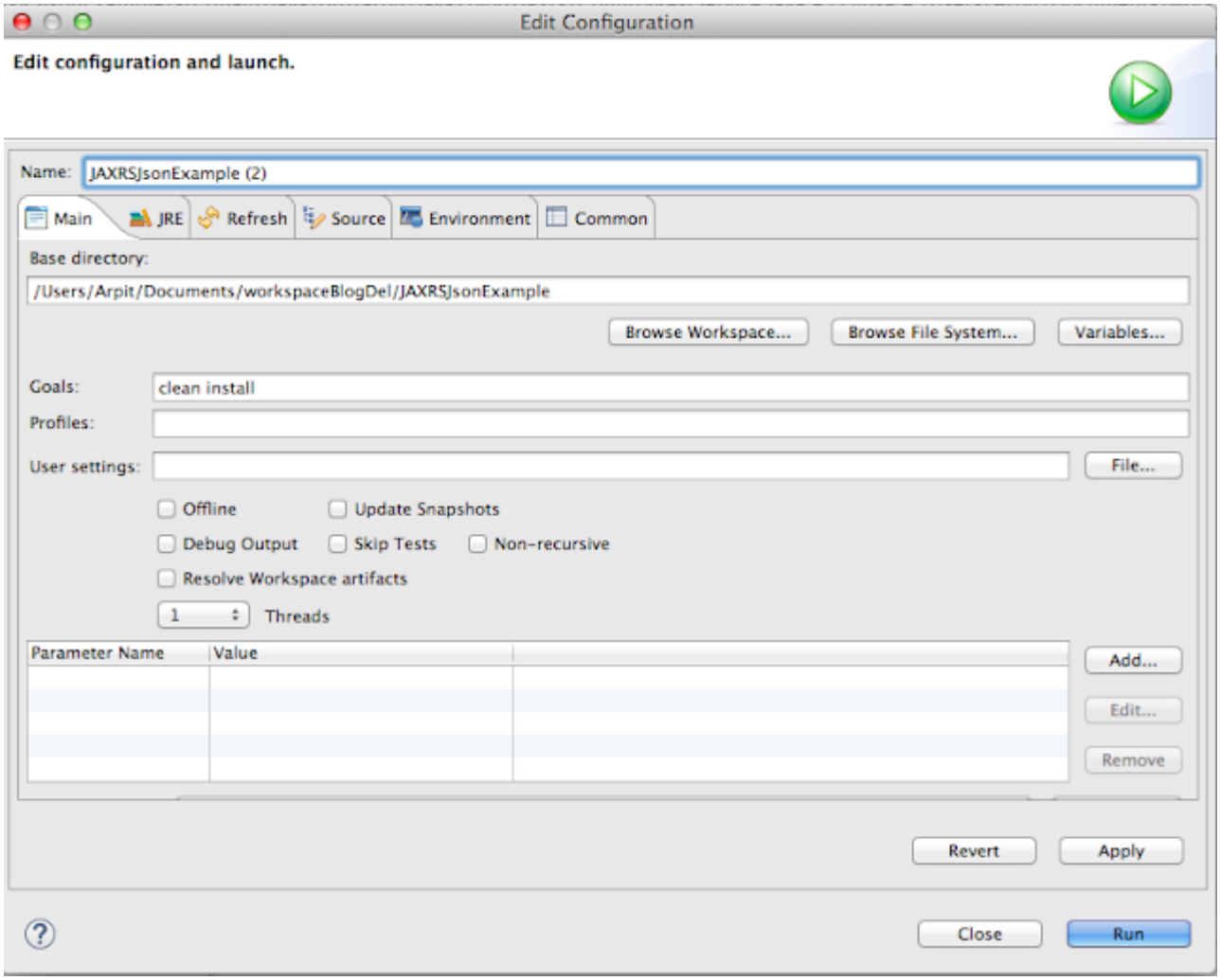
```
11  * It is just a helper class which should be replaced by database implementation.
12  * It is not very well written class, it is just used for demonstration.
13  */
14  public class CountryService {
15
16   static HashMap<Integer,Country> countryIdMap=getCountryIdMap();
17
18
19   public CountryService() {
20    super();
21
22    if(countryIdMap==null)
23    {
24     countryIdMap=new HashMap<Integer,Country>();
25    // Creating some objects of Country while initializing
26     Country indiaCountry=new Country(1, "India",10000);
27     Country chinaCountry=new Country(4, "China",20000);
28     Country nepalCountry=new Country(3, "Nepal",8000);
29     Country bhutanCountry=new Country(2, "Bhutan",7000);
30
31
32     countryIdMap.put(1,indiaCountry);
33     countryIdMap.put(4,chinaCountry);
34     countryIdMap.put(3,nepalCountry);
35     countryIdMap.put(2,bhutanCountry);
36    }
37   }
38
39   public List getAllCountries()
40   {
41    List countries = new ArrayList(countryIdMap.values());
42    return countries;
43   }
44
45   public Country getCountry(int id)
```

```
46  {
47   Country country= countryIdMap.get(id);
48   return country;
49  }
50  public Country addCountry(Country country)
51  {
52   country.setId(countryIdMap.size()+1);
53   countryIdMap.put(country.getId(), country);
54   return country;
55  }
56
57  public Country updateCountry(Country country)
58  {
59   if(country.getId()<=0)
60    return null;
61   countryIdMap.put(country.getId(), country);
62   return country;
63
64  }
65  public void deleteCountry(int id)
66  {
67   countryIdMap.remove(id);
68  }
69
70  public static HashMap<Integer, Country> getCountryIdMap() {
71   return countryIdMap;
72  }
73 }
74
```

7) It ‘s time to do maven build.
Right click on project -> Run as -> Maven build



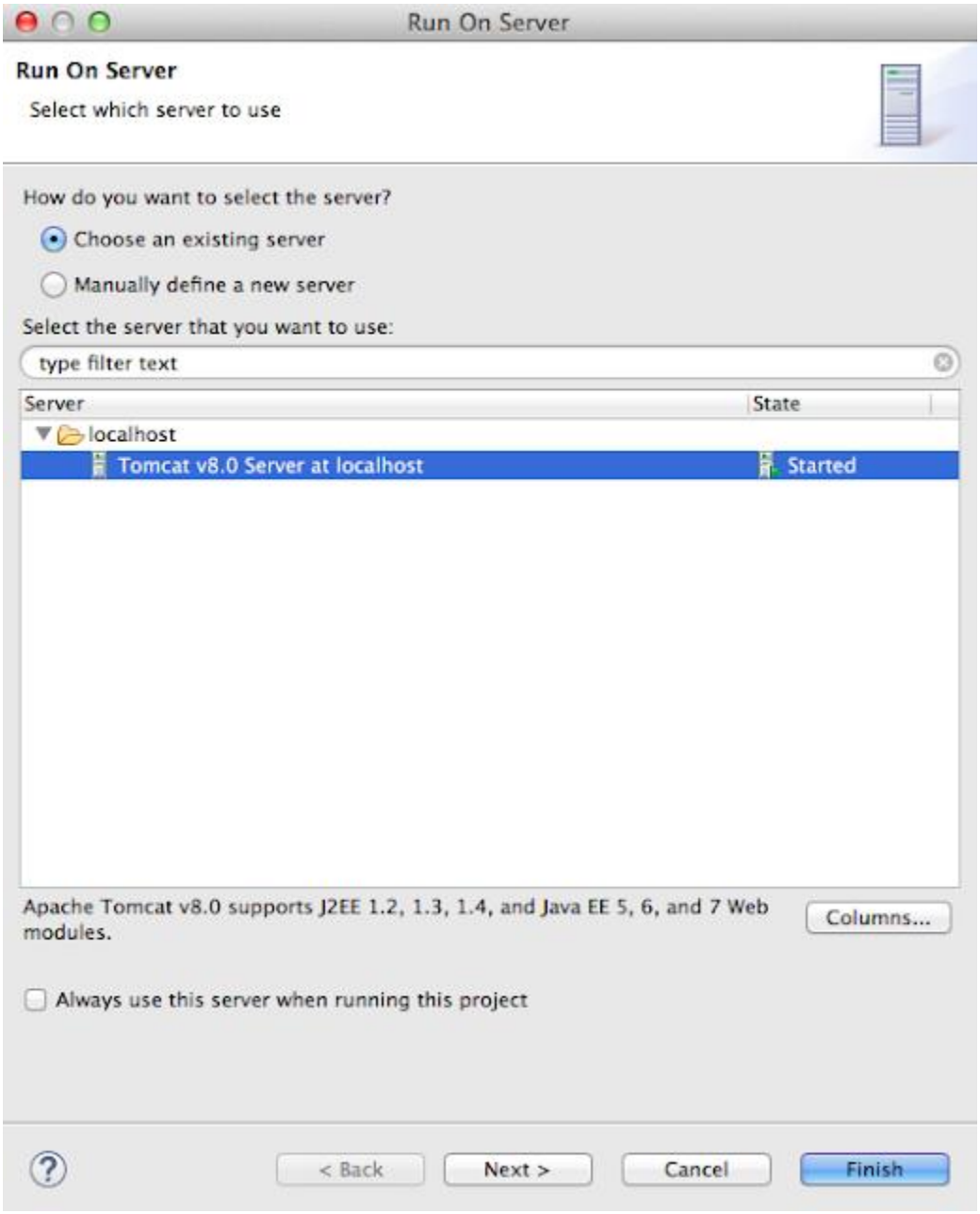
8) Provide goals as clean install (given below) and click on run



Run the application

9) Right click on project -> run as -> run on server

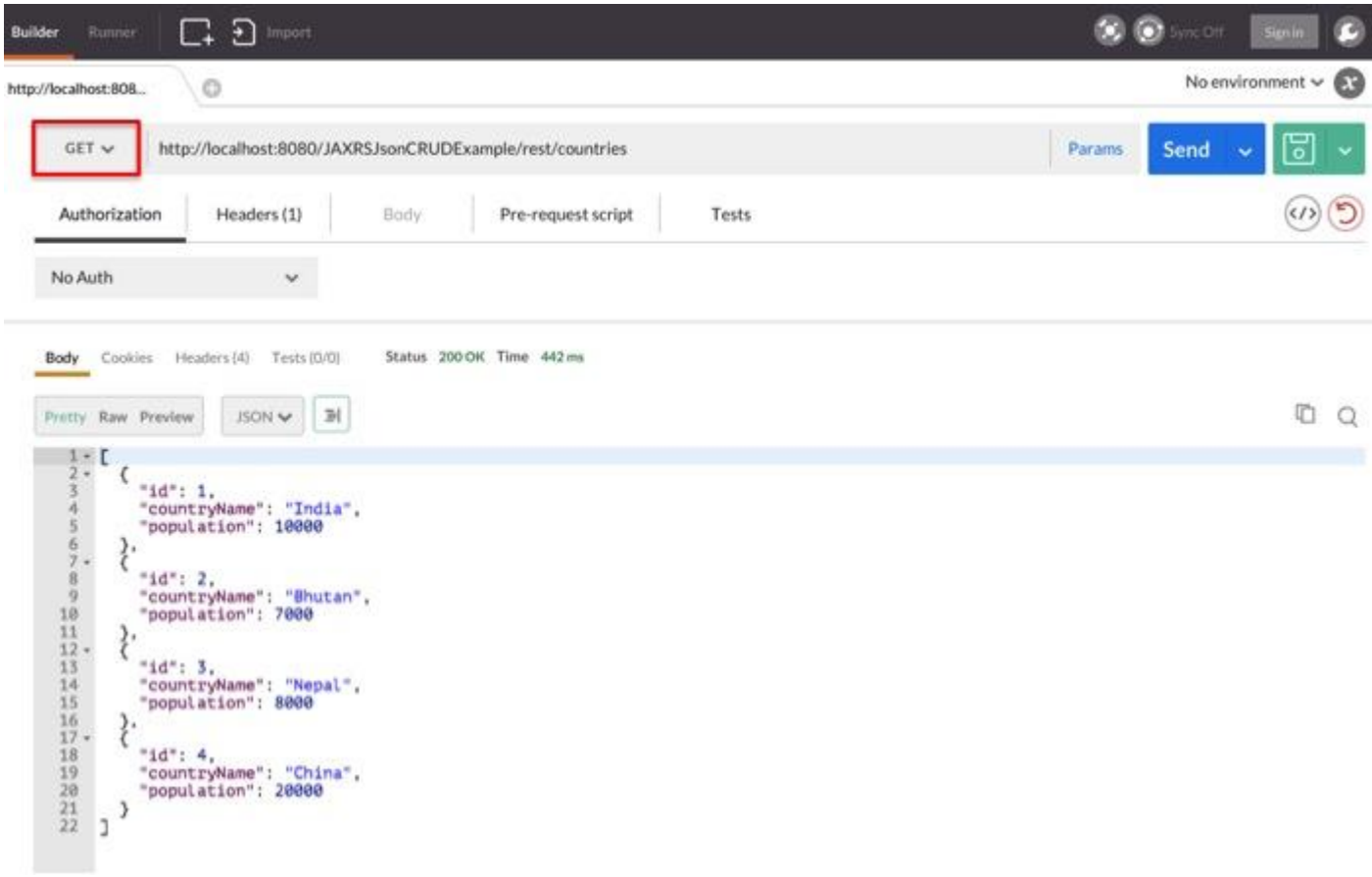
Select apache tomcat and click on finish



10) We will test this application in postman , UI based client for testing restful web applications. It is chrome plugin. Launch postman. If you want java based client, then you can also use how to send get or post request in java.

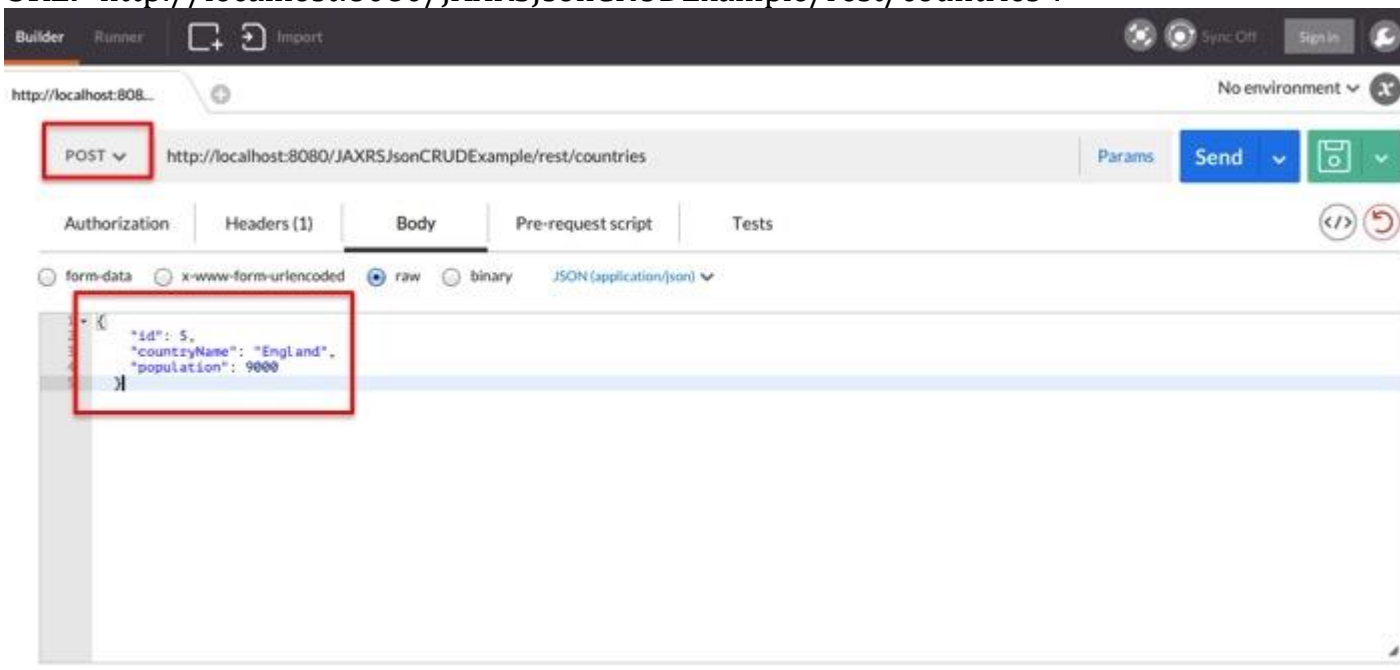
Get method

11) Test your get method REST service **URL :**"http://localhost:8080/JAXRSJsonCRUDExample/rest/countries".
You will get following output:

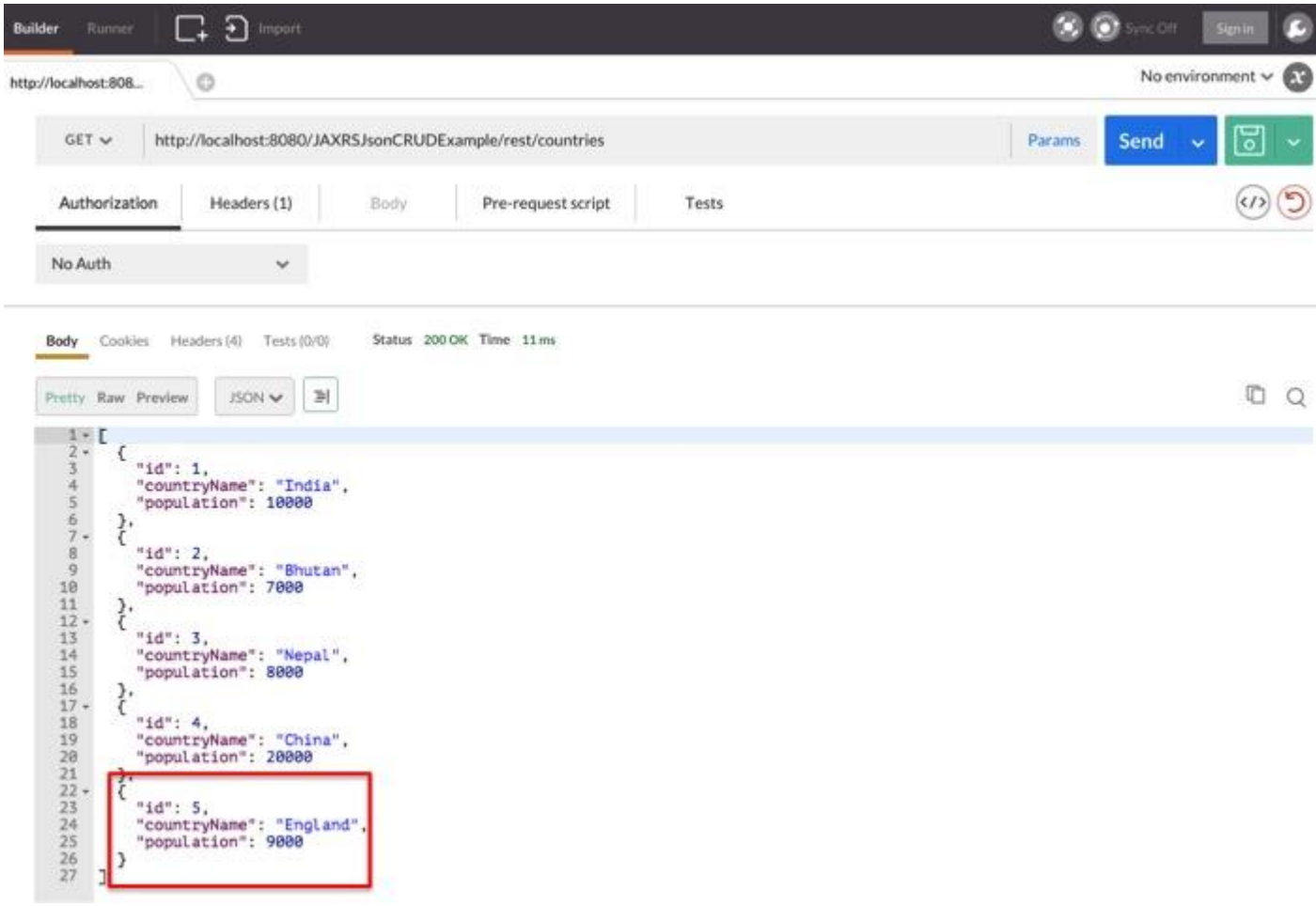


Post method

12) Post method is used to create new resource. Here we are adding new Country England to country list, so you can see we have used new country json in post body.
URL: "http://localhost:8080/JAXRSJsonCRUDExample/rest/countries".

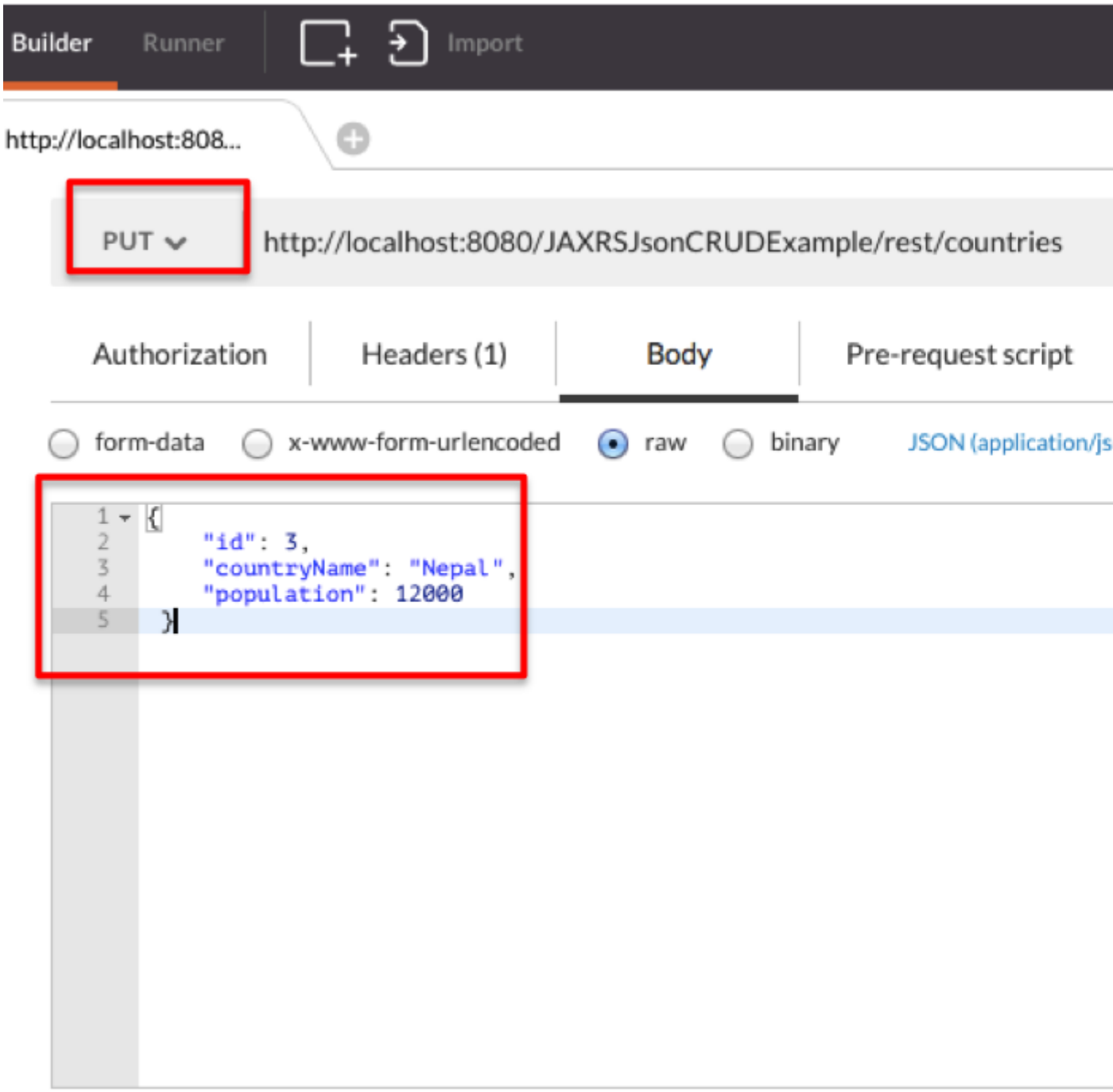


Use get method to check if above country have been added to country list.

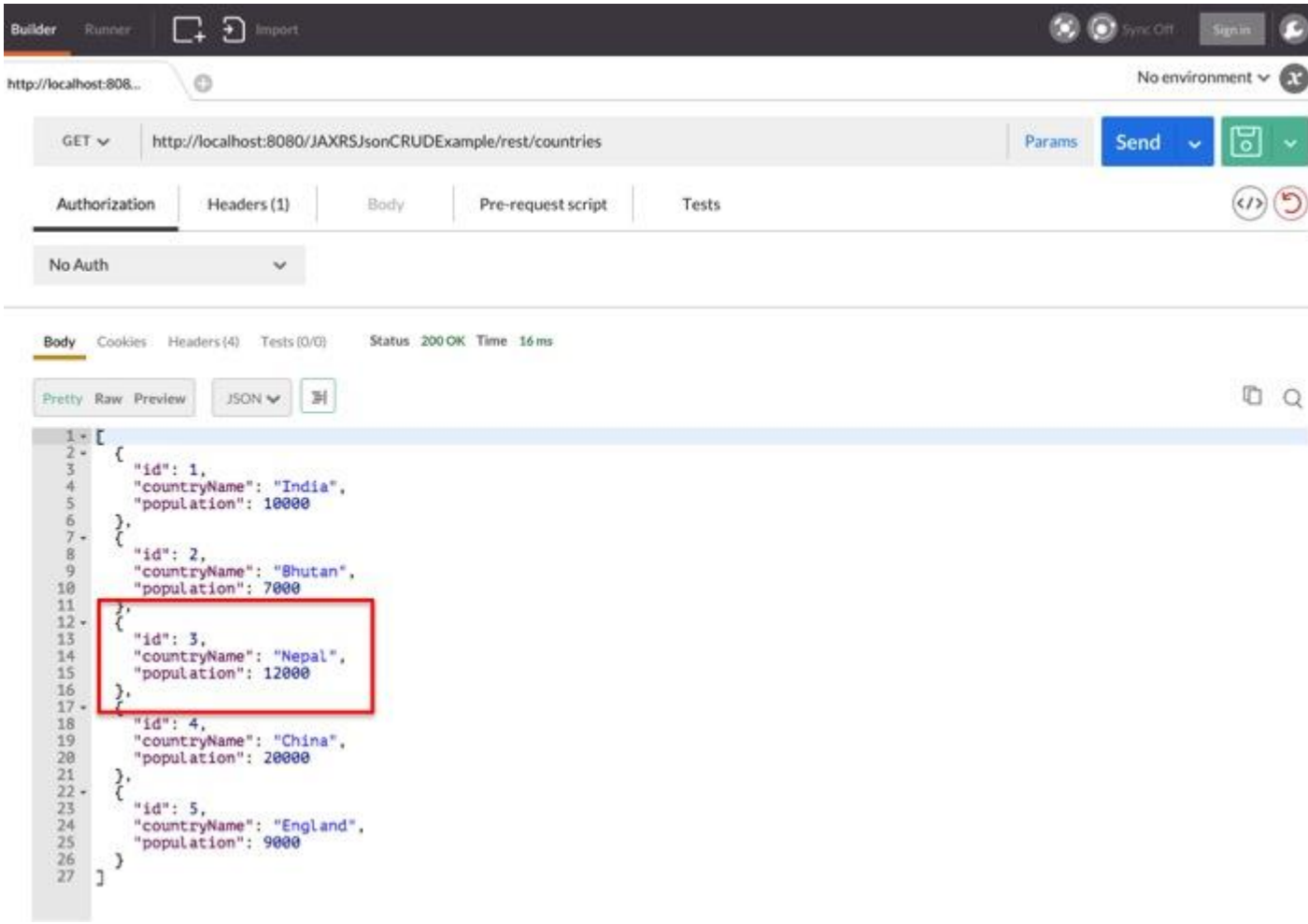


Put Method

13) Put method is used to update resource. Here will update population of nepal using put method.
We will update country json in body of request.
URL : "http://localhost:8080/JAXRSJsonCRUDExample/rest/countries"



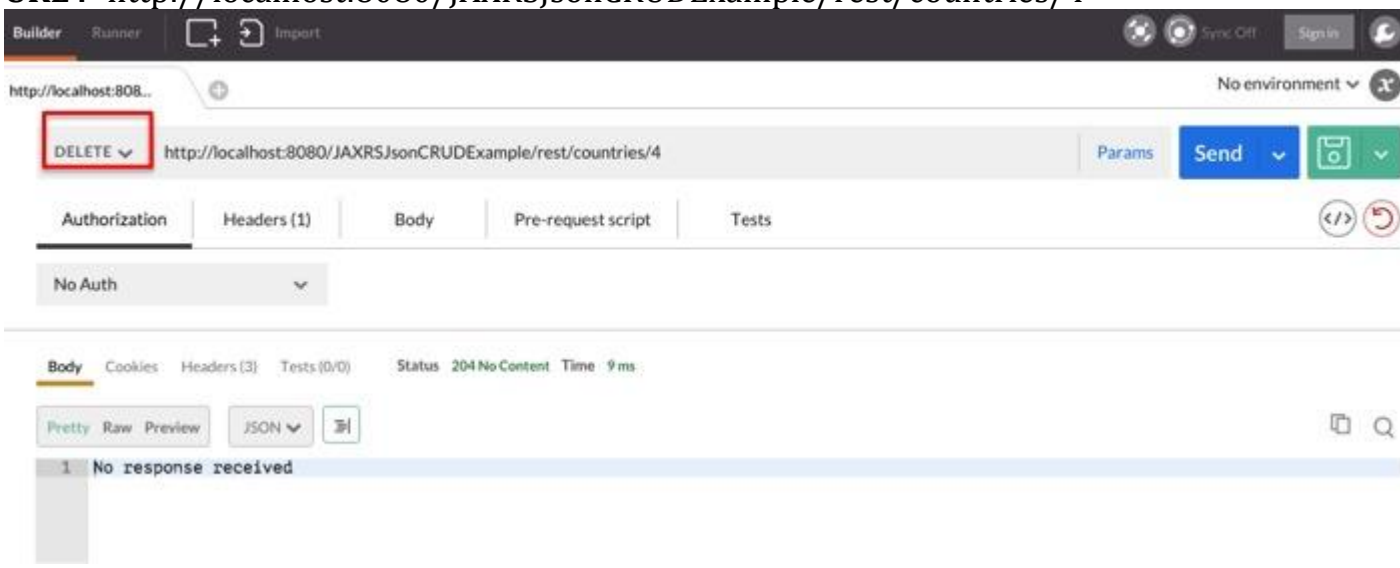
Use get method to check population of nepal.



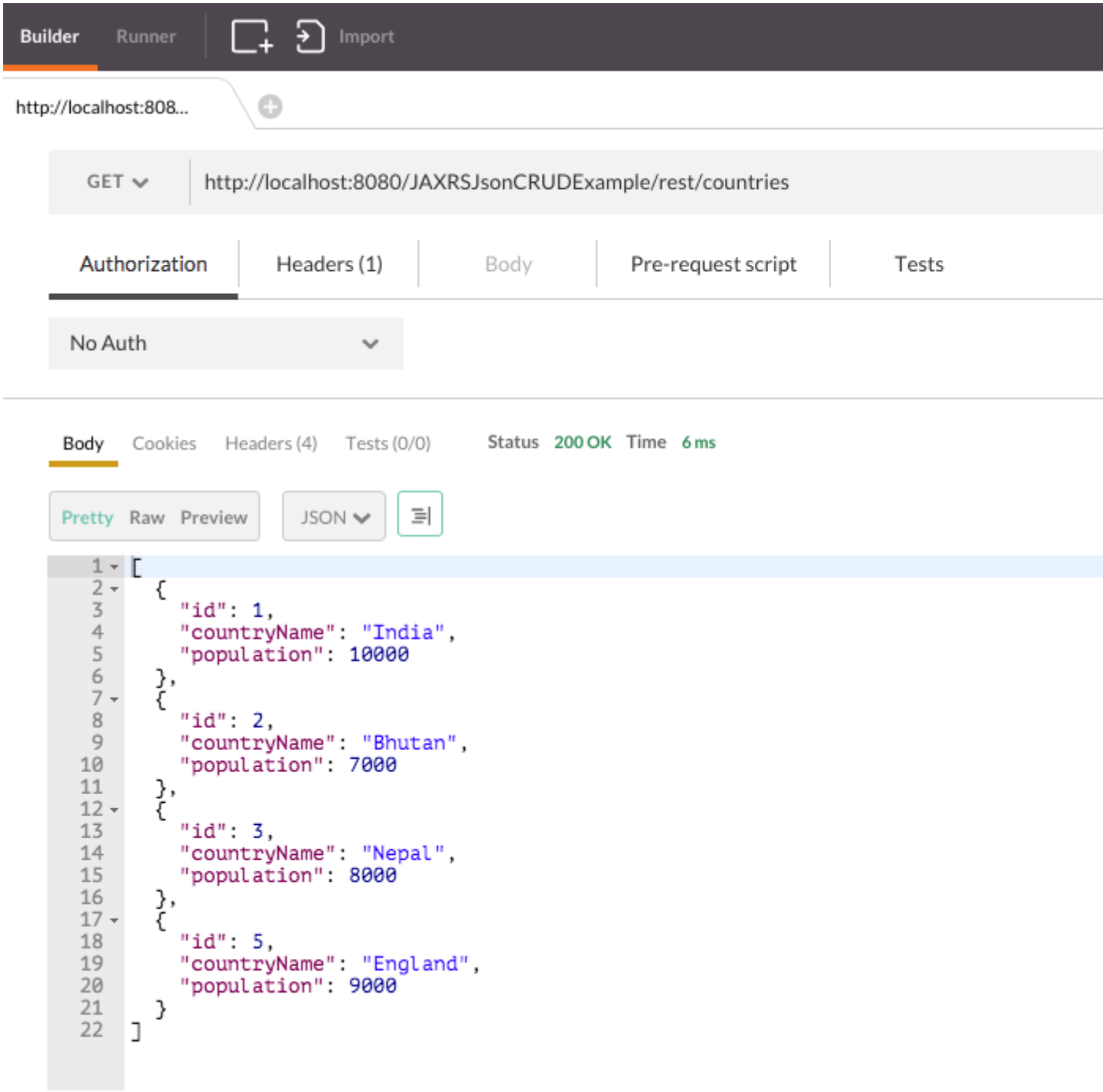
Delete method

14) Delete method is used to delete resource.We will pass id of country which needs to be deleted as PathParam. We are going delete id:4 i.e. china to demonstrate delete method.

URL : "http://localhost:8080/JAXRSJsonCRUDExample/rest/countries/4"

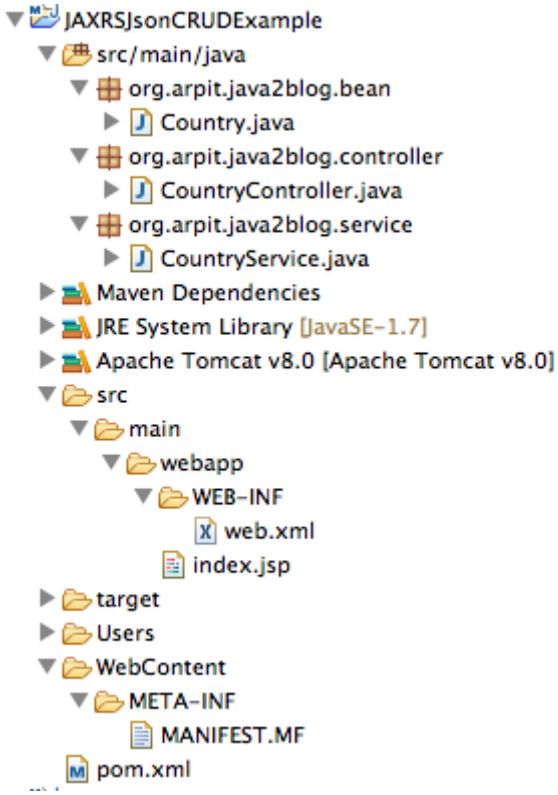


Use get method to check country list.



As you can see, we have deleted **country with id 4 i.e. china**

Project structure:



We are done with Restful web services json CRUD example using jersey

Spring Restful web services example::

We have already seen Restful web services tutorial and Spring mvc web example. In this post, we will see combination of both.
Spring have started supporting Restful web services after Spring 3 version. In this post, we will see how to create Restful web services with Spring framework.If you want complete integration with hibernate and mysql, you can go through Spring Restful hibernate mysql example.

Here are steps to create a simple Spring Restful web services which will return plain text.

1) Create a dynamic web project using maven in eclipse.
2) Now change pom.xml as follows:
pom.xml

```
1
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
ance"
3 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.arpit.java2blog</groupId>
<artifactId>SpringRestfulWebServicesExample</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringRestfulWebServicesExample Maven Webapp</name>
<url>http://maven.apache.org</url>
<dependencies>
 <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
 </dependency>
 <dependency>
<groupId>javax.servlet</groupId>
<artifactId>javax.servlet-api</artifactId>
<version>3.1.0</version>
</dependency>

<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-core</artifactId>
 <version>${spring.version}</version>
</dependency>
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-webmvc</artifactId>
 <version>${spring.version}</version>
</dependency>
</dependencies>
<build>
 <finalName>SpringRestfulHelloWorldExample</finalName>

 <plugins>
```

```
39   <plugin>
40    <groupId>org.apache.maven.plugins</groupId>
41    <artifactId>maven-compiler-plugin</artifactId>
42    <version>3.1</version>
43    <configuration>
44     <source>${jdk.version}</source>
45     <target>${jdk.version}</target>
46    </configuration>
47   </plugin>
48  </plugins>
49
50 </build>
51 <properties>
52  <spring.version>4.2.1.RELEASE</spring.version>
53  <jdk.version>1.7</jdk.version>
54 </properties>
55
56 </project>
57
```

Spring application configuration:

3) Change web.xml as below:

```
 1
 2  <!DOCTYPE web-app PUBLIC
 3   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 4   "http://java.sun.com/dtd/web-app_2_3.dtd" >
 5
 6  <web-app>
 7   <display-name>Archetype Created Web Application</display-name>
 8   <servlet>
 9   <servlet-name>springrest</servlet-name>
10   <servlet-class>
11    org.springframework.web.servlet.DispatcherServlet
12   </servlet-class>
13   <load-on-startup>1</load-on-startup>
```

```
</servlet>

<servlet-mapping>
 <servlet-name>springrest</servlet-name>
 <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>

```

4) create a xml file named springrest-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to spring mvc hello world example for more understanding.**

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

 <mvc:annotation-driven/>
<context:component-scan base-package="org.arpit.java2blog.controller" />

</beans>
```

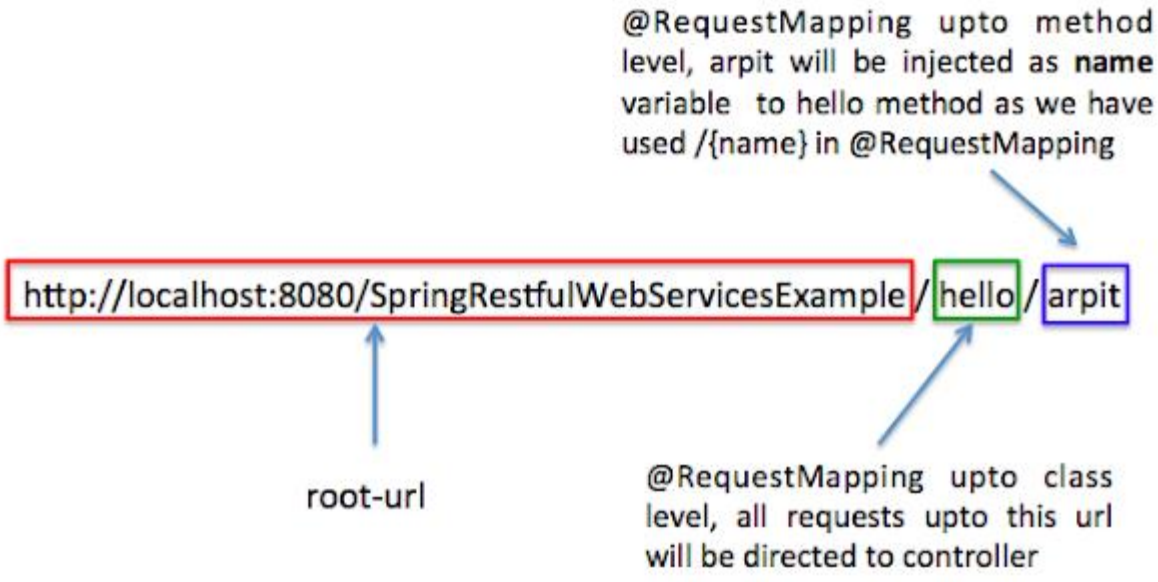
Create controller

5) Create a controller named “SpringRestController.java”

```

package org.arpit.java2blog.controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;


```

```
@RestController
@RequestMapping("/hello")
public class SpringRestController {
	@RequestMapping(value = "/{name}", method = RequestMethod.GET)
	public String hello(@PathVariable String name) {
		String result="Hello "+name;
		return result;
	}
}
```

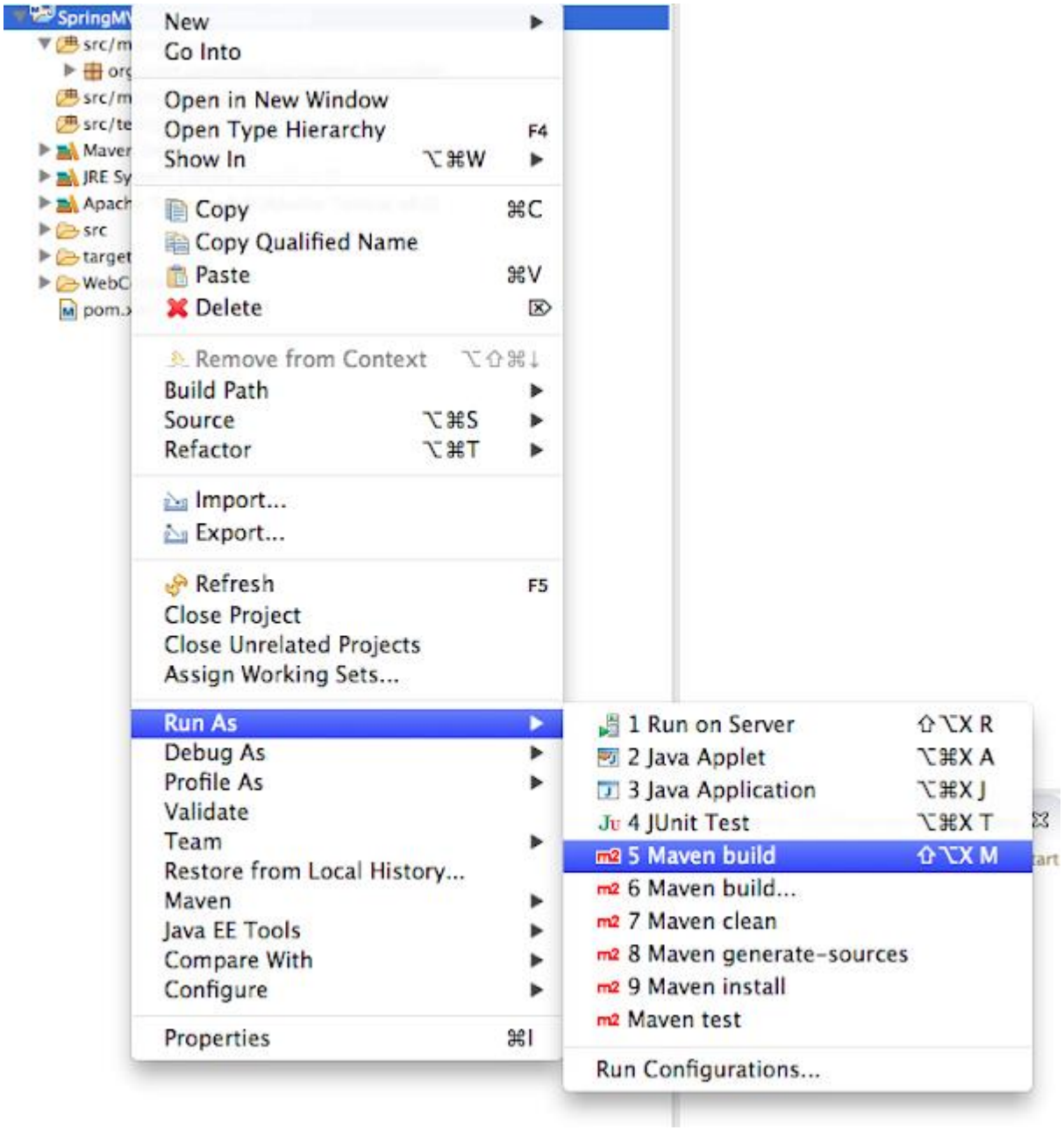
@PathVariable: Used to inject values from the URL into a method parameter.This way you inject name in hello method .



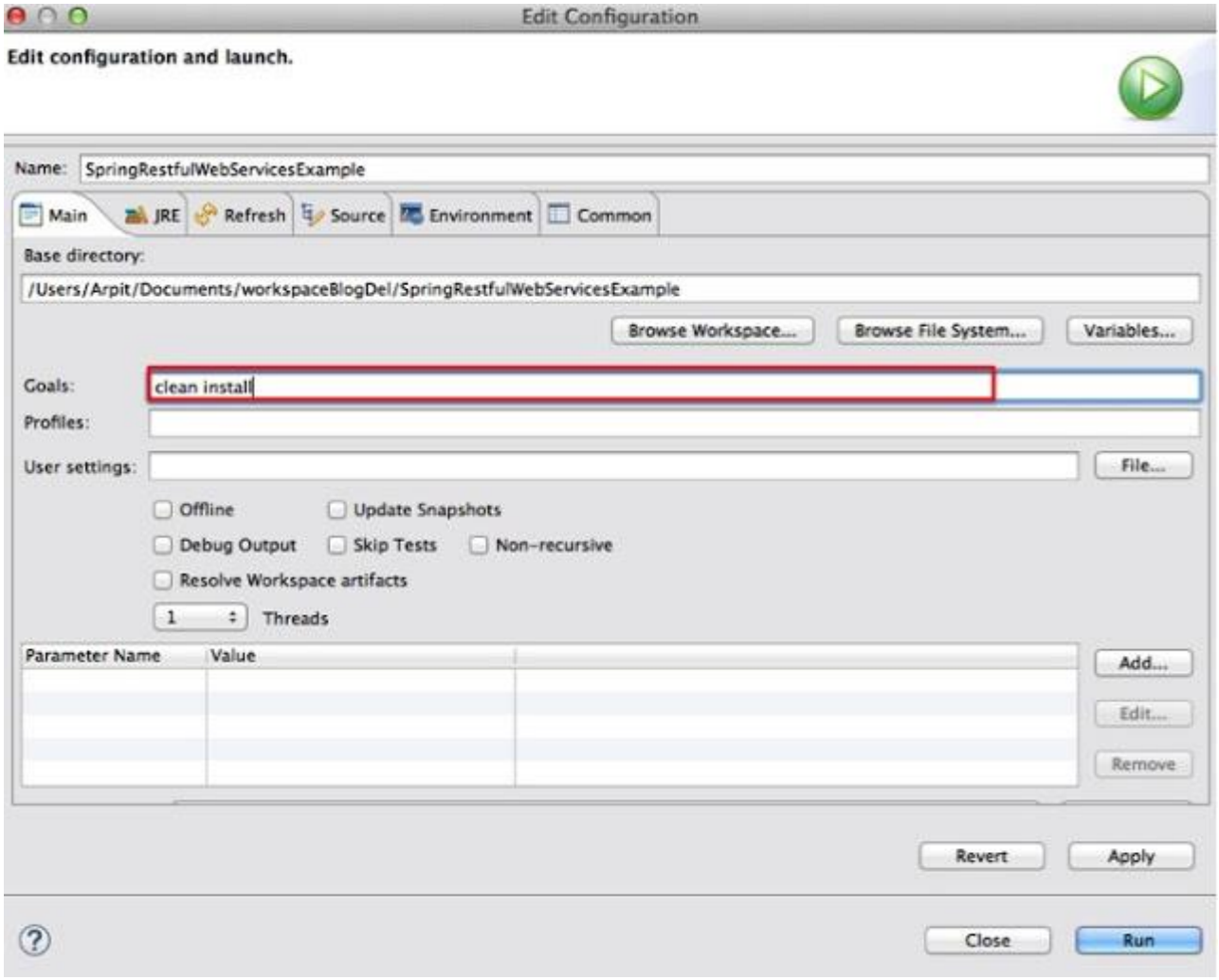
We are not providing any view information in springrest-servlet.xml as we do in Spring MVC. If we need to directly get resource from controller, we need to return @ResponseBody as per Spring 3 but with Spring 4, we can use @RestController for that.
In spring 4.0, we can use @RestController which is combination of @Controller + @ResponseBody.

```
@RestController = @Controller + @ResponseBody
```

6) It 's time to do maven build.
Right click on project -> Run as -> Maven build



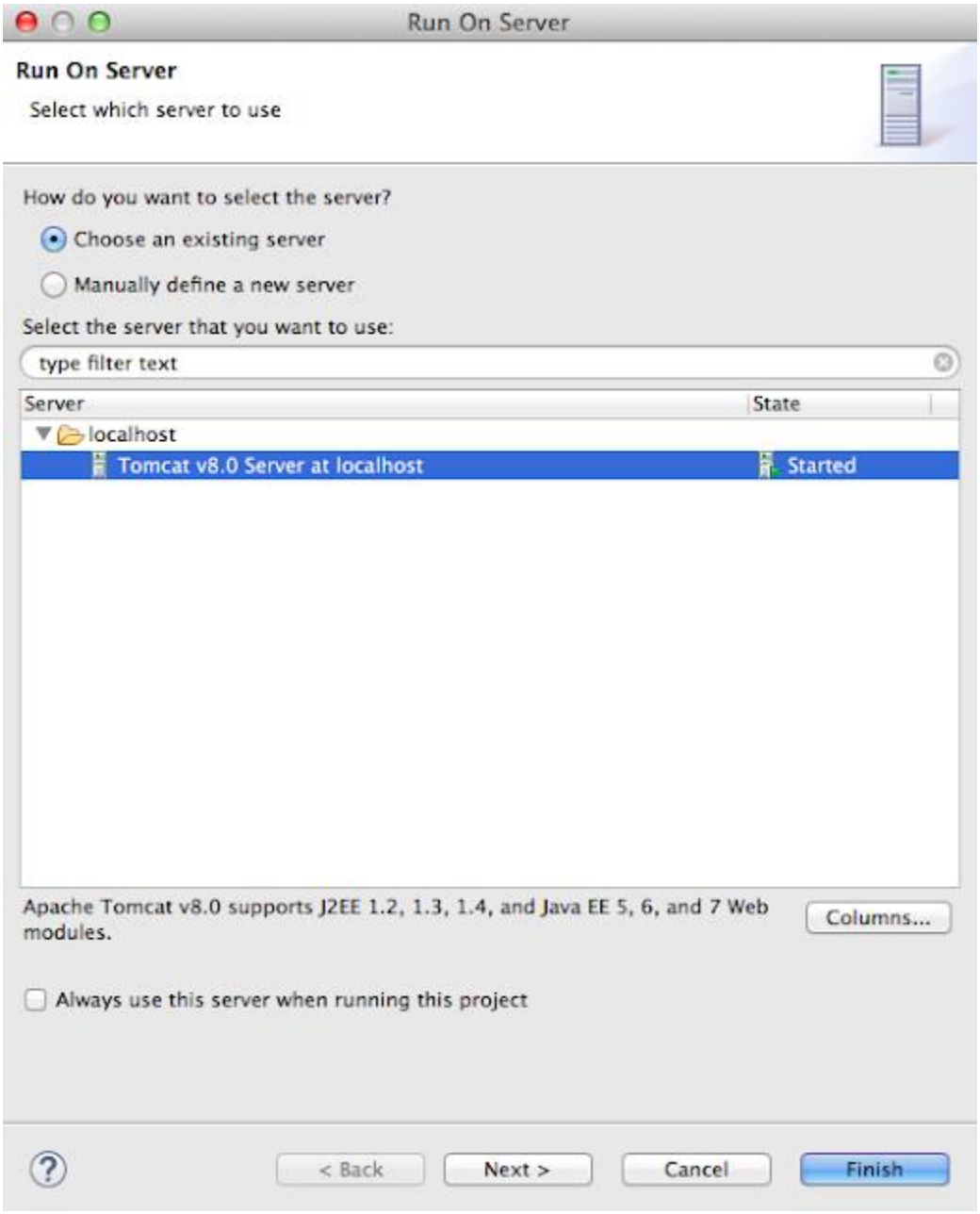
7) Provide goals as clean install (given below) and click on run



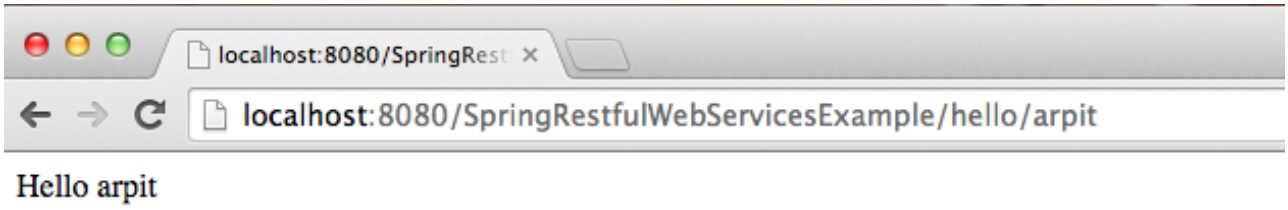
Run the application

8) Right click on project -> run as -> run on server

Select apache tomcat and click on finish



9) Test your REST service under: "http://localhost:8080/SpringRestfulWebServicesExample/hello/arpit". You will get following output:



Download
click to begin

20KB .zip

We are done with Spring Restful web services example

Spring Restful web services json example::

In previous post, we have created a very simple Spring Restful web services which returns plain text. In this post, we will see Spring Restful web services which will return json as example. If you want complete integration with hibernate and mysql, you can go through Spring Restful hibernate mysql example.
Here are steps to create a simple Spring Restful web services which will return json.
1) Create a dynamic web project using maven in eclipse.
2) We need to add Jackson json utility in the classpath.

```
1
2
  <dependency>
3
        <groupId>com.fasterxml.jackson.core</groupId>
4
        <artifactId>jackson-databind</artifactId>
5
         <version>2.4.1</version>
6
   </dependency>
7
```

Spring will load Jackson2JsonMessageConverter into its application context automatically. Whenever you request resource as json with

accept headers=”Accept=application/json”, then Jackson2JsonMessageConverter comes into picture and convert resource to json format.

Now change pom.xml as follows:

pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.arpit.java2blog</groupId>
6   <artifactId>SpringRestfulWebServicesWithJSONExample</artifactId>
7   <packaging>war</packaging>
8   <version>0.0.1-SNAPSHOT</version>
9   <name>SpringRestfulWebServicesWithJSONExample Maven Webapp</name>
10  <url>http://maven.apache.org</url>
11  <dependencies>
12   <dependency>
13    <groupId>junit</groupId>
14    <artifactId>junit</artifactId>
15    <version>3.8.1</version>
16    <scope>test</scope>
17   </dependency>
18   <dependency>
19  <groupId>javax.servlet</groupId>
20  <artifactId>javax.servlet-api</artifactId>
21  <version>3.1.0</version>
22  </dependency>
23
24 <dependency>
25  <groupId>org.springframework</groupId>
26  <artifactId>spring-core</artifactId>
27  <version>${spring.version}</version>
28 </dependency>
29 <dependency>
30  <groupId>org.springframework</groupId>
31
```

```
32   <artifactId>spring-webmvc</artifactId>
33   <version>${spring.version}</version>
34  </dependency>
35   <dependency>
36          <groupId>com.fasterxml.jackson.core</groupId>
37          <artifactId>jackson-databind</artifactId>
38           <version>2.4.1</version>
39       </dependency>
40 </dependencies>
41 <build>
42  <finalName>SpringRestfulWebServicesWithJSONExample</finalName>
43
44  <plugins>
45   <plugin>
46    <groupId>org.apache.maven.plugins</groupId>
47    <artifactId>maven-compiler-plugin</artifactId>
48    <version>3.1</version>
49    <configuration>
50     <source>${jdk.version}</source>
51     <target>${jdk.version}</target>
52    </configuration>
53   </plugin>
54  </plugins>
55
56 </build>
57 <properties>
58  <spring.version>4.2.1.RELEASE</spring.version>
59  <jdk.version>1.7</jdk.version>
60 </properties>
61 </project>
```

Spring application configuration:

3) Change web.xml as below:

```
1
2  <!DOCTYPE web-app PUBLIC
```

```
3  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4  "http://java.sun.com/dtd/web-app_2_3.dtd" >
5
6  <web-app>
7   <display-name>Archetype Created Web Application</display-name>
8   <servlet>
9  <servlet-name>springrest</servlet-name>
10 <servlet-class>
11  org.springframework.web.servlet.DispatcherServlet
12 </servlet-class>
13 <load-on-startup>1</load-on-startup>
14 </servlet>
15
16 <servlet-mapping>
17  <servlet-name>springrest</servlet-name>
18  <url-pattern>/</url-pattern>
19 </servlet-mapping>
20 </web-app>
21
```

4) create a xml file named springrest-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to [spring mvc hello world example](#) for more understanding.**

```
1
2  <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:context="http://www.springframework.org/schema/context"
4   xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSche
   ma-instance"
5   xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/sch
6  ema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context
7        http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.o
   rg/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
8
9   <mvc:annotation-driven/>
10 <context:component-scan base-package="org.arpit.java2blog.controller" />
11
12 </beans>
```

Create bean class

5) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```
1
2  package org.arpit.java2blog.bean;
3
4  public class Country{
5
6          int id;
7          String countryName;
8
9          public Country(int i, String countryName) {
10                 super();
11                 this.id = i;
12                 this.countryName = countryName;
13         }
14         public int getId() {
15                 return id;
16         }
17         public void setId(int id) {
18                 this.id = id;
19         }
20         public String getCountryName() {
21                 return countryName;
22         }
23         public void setCountryName(String countryName) {
24                 this.countryName = countryName;
25         }
26
27 }
28
```

Create controller

6) Create a controller named “CountryController.java”

```
1
2  package org.arpit.java2blog.controller;
```

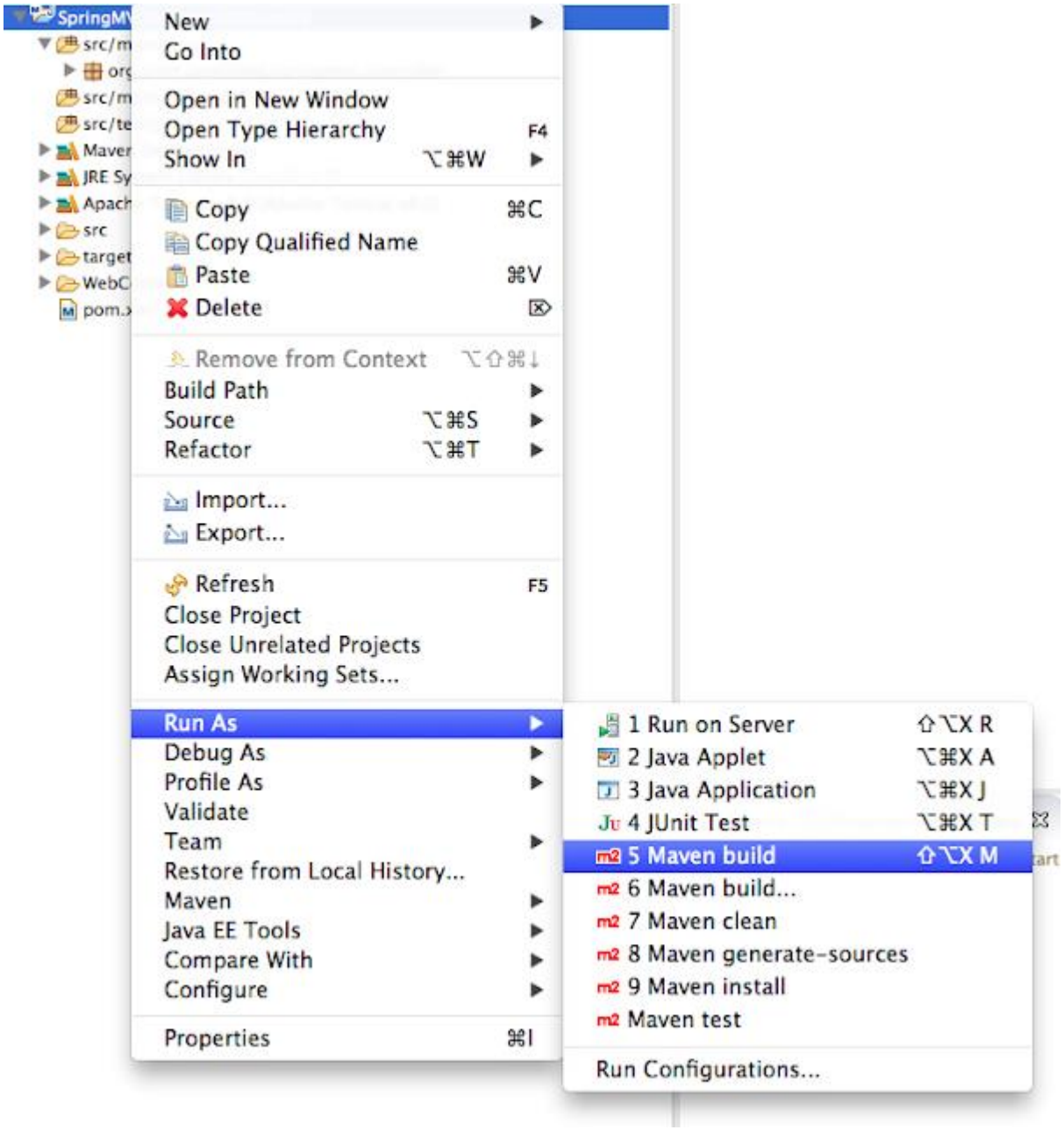
```
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import org.arpit.java2blog.bean.Country;
8  import org.springframework.web.bind.annotation.PathVariable;
9  import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11 import org.springframework.web.bind.annotation.RestController;
12
13 @RestController
14 public class CountryController {
15
16          @RequestMapping(value = "/countries", method = RequestMethod.GET,headers="Accept=application/j
   son")
17
18          public List getCountries()
19          {
20                  List listOfCountries = new ArrayList();
21                  listOfCountries=createCountryList();
22                  return listOfCountries;
23          }
24
25          @RequestMapping(value = "/country/{id}", method = RequestMethod.GET,headers="Accept=applicatio
   n/json")
26          public Country getCountryById(@PathVariable int id)
27          {
28                  List listOfCountries = new ArrayList();
29                  listOfCountries=createCountryList();
30
31                  for (Country country: listOfCountries) {
32                          if(country.getId()==id)
33                                  return country;
34                  }
35
36                  return null;
37          }
```

```
38
39          // Utiliy method to create country list.
40          public List createCountryList()
41          {
42                  Country indiaCountry=new Country(1, "India");
43                  Country chinaCountry=new Country(4, "China");
44                  Country nepalCountry=new Country(3, "Nepal");
45                  Country bhutanCountry=new Country(2, "Bhutan");
46
47                  List listOfCountries = new ArrayList();
48                  listOfCountries.add(indiaCountry);
49                  listOfCountries.add(chinaCountry);
50                  listOfCountries.add(nepalCountry);
51                  listOfCountries.add(bhutanCountry);
52                  return listOfCountries;
53          }
54 }
```

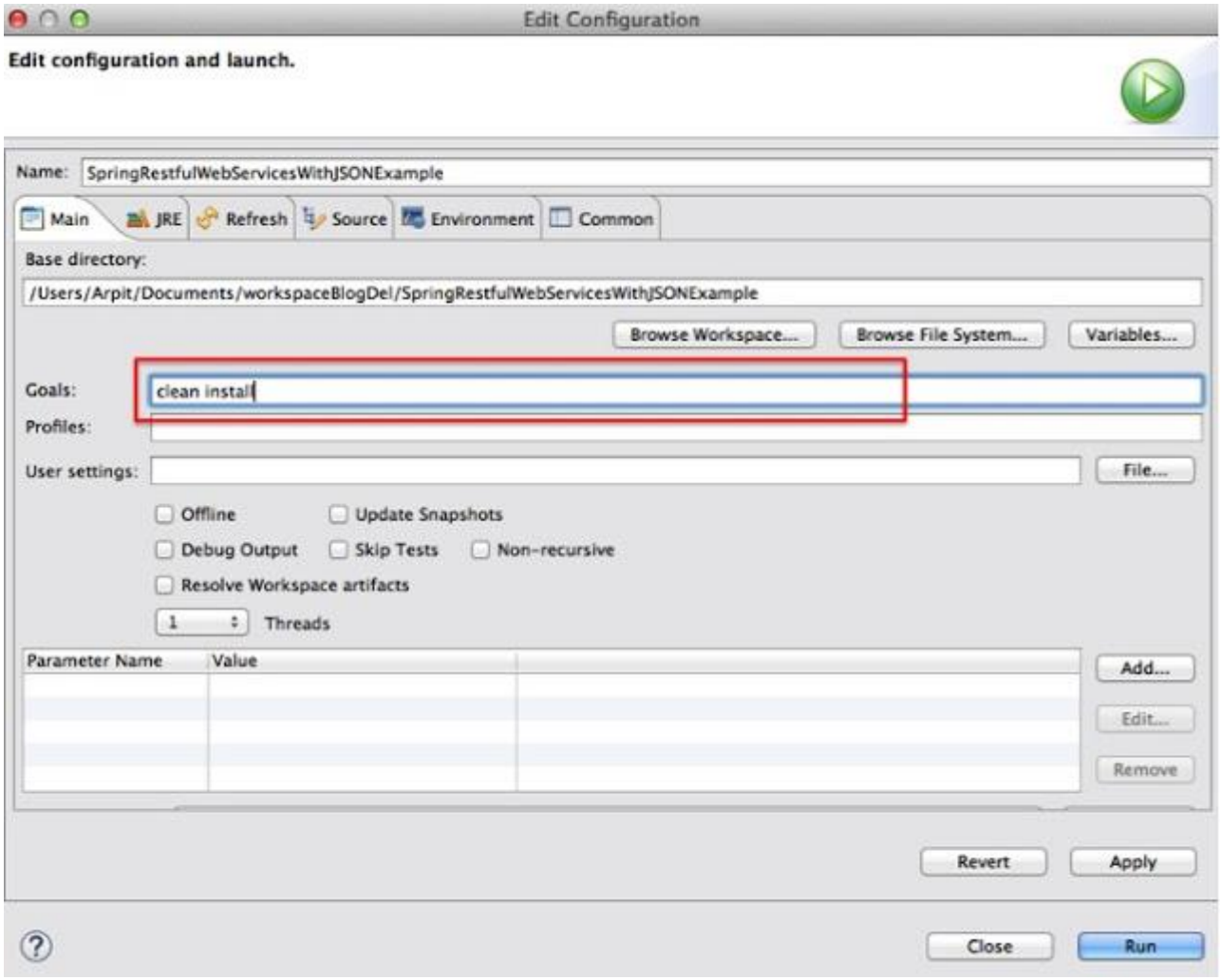
@PathVariable: Used to inject values from the URL into a method parameter.This way you inject id in getCountryById method.
We are not providing any view information in springrest-servlet.xml as we do in Spring MVC. If we need to directly get resource from controller, we need to return @ResponseBody as per Spring 3 but with Spring 4, we can use @RestController for that.
In spring 4.0, we can use @RestController which is combination of @Controller + @ResponseBody.

```
1
2 @RestController = @Controller + @ResponseBody
3
```

6) It 's time to do maven build.
Right click on project -> Run as -> Maven build



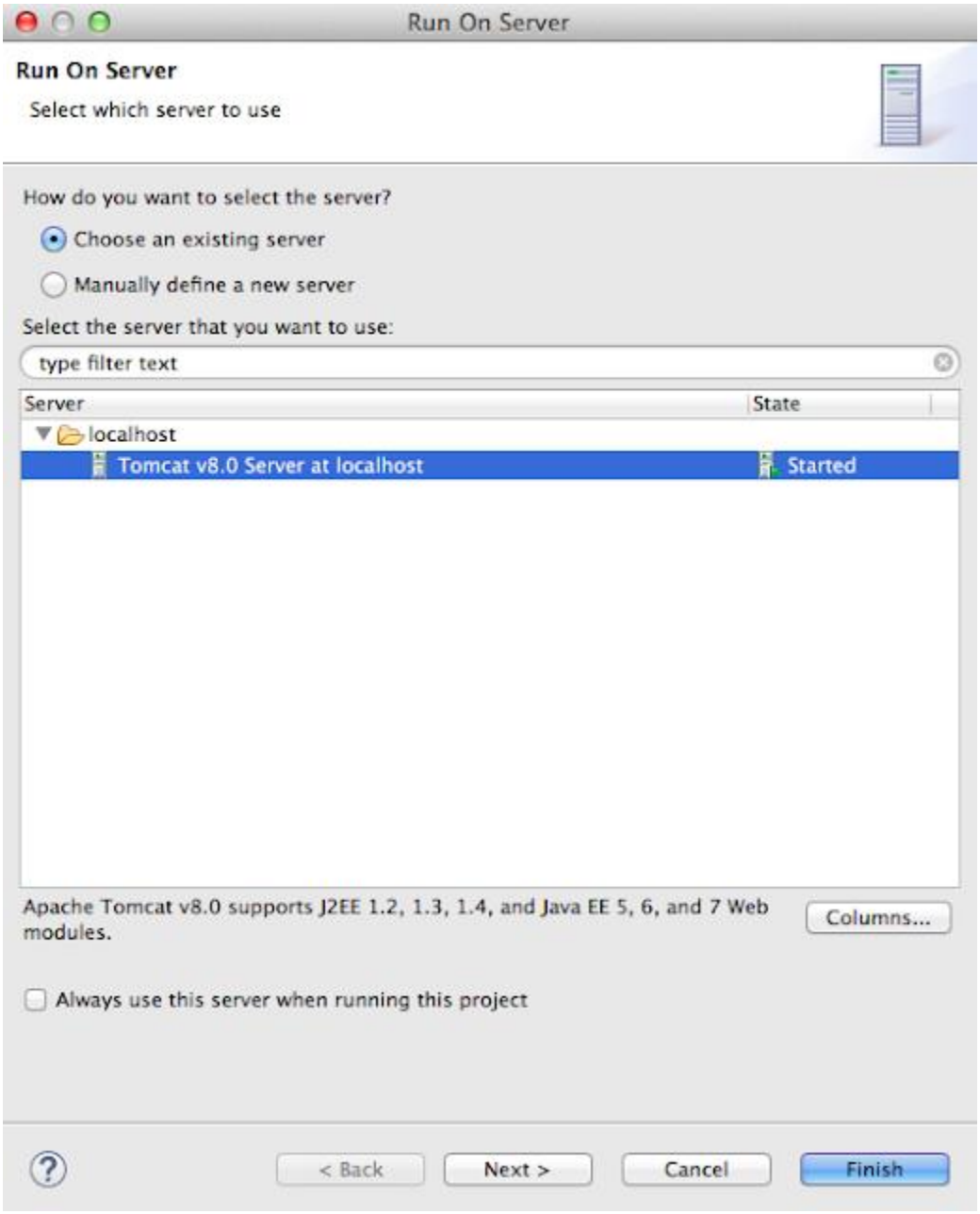
7) Provide goals as clean install (given below) and click on run



Run the application

8) Right click on project -> run as -> run on server

Select apache tomcat and click on finish



When you run the application, you might get this kind of warning

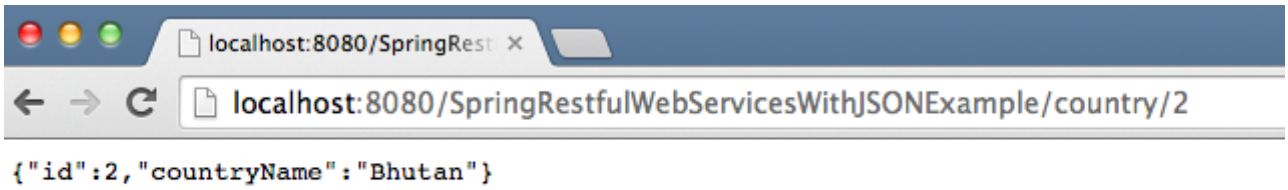
```
1
2 Mar 26, 2016 1:45:51 AM org.springframework.web.servlet.PageNotFound noHandlerFound
3 WARNING: No mapping found for HTTP request with URI [/SpringRestfulWebServicesWithJSONExample/] in Disp
  atcherServlet with name 'SpringRestfulWebServicesWithJSONExample'
4
```

Please ignore above warning. When you start application, you have below URL if you have not provided start page: **http://localhost:8080/SpringRestfulWebServicesWithJSONExample/**

As we have used DispatcherServlet in web.xml, this request goes to spring DispatcherServlet and it did not find corresponding mapping in controller , hence you get that warning.

9) Test your REST service under: "http://localhost:8080/SpringRestfulWebServicesWithJSONExample/countries". You will get following output:

10) Now pass country id as parameter to url. "http://localhost:8080/SpringRestfulWebServicesWithJSONExample/country/2".



Download

click to begin

20KB .zip

We are done with Spring Restful web services Json example.

Spring Restful web services xml example::

In previous post, we have created a very simple Spring Restful web services which returns json. In this post, we will see Spring Restful web services which will return xml as example.
Here are steps to create a simple Spring Restful web services which will return xml.

1) Create a dynamic web project using maven in eclipse.
2) For XML support, we just need to make sure JAXB jar is available in classpath.

pom.xml will be as follows:

pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.arpit.java2blog</groupId>
6   <artifactId>SpringRestfulWebServicesWithJSONExample</artifactId>
7   <packaging>war</packaging>
8   <version>0.0.1-SNAPSHOT</version>
9   <name>SpringRestfulWebServicesWithJSONExample Maven Webapp</name>
10  <url>http://maven.apache.org</url>
11  <dependencies>
12   <dependency>
13    <groupId>junit</groupId>
14    <artifactId>junit</artifactId>
15    <version>3.8.1</version>
16    <scope>test</scope>
17   </dependency>
18   <dependency>
19  <groupId>javax.servlet</groupId>
20   <artifactId>javax.servlet-api</artifactId>
21   <version>3.1.0</version>
22  </dependency>
23
24  <dependency>
25   <groupId>org.springframework</groupId>
26   <artifactId>spring-core</artifactId>
27   <version>${spring.version}</version>
28  </dependency>
29  <dependency>
30   <groupId>org.springframework</groupId>
31   <artifactId>spring-webmvc</artifactId>
32   <version>${spring.version}</version>
33  </dependency>
34
```

```
35
36 </dependencies>
37 <build>
38  <finalName>SpringRestfulWebServicesWithJSONExample</finalName>
39
40  <plugins>
41   <plugin>
42    <groupId>org.apache.maven.plugins</groupId>
43    <artifactId>maven-compiler-plugin</artifactId>
44    <version>3.1</version>
45    <configuration>
46     <source>${jdk.version}</source>
47     <target>${jdk.version}</target>
48    </configuration>
49   </plugin>
50  </plugins>
51
52 </build>
53 <properties>
54  <spring.version>4.2.1.RELEASE</spring.version>
55  <jdk.version>1.7</jdk.version>
56 </properties>
57 </project>
```

Spring application configuration:

3) Change web.xml as below:

```
1
2  <!DOCTYPE web-app PUBLIC
3   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4   "http://java.sun.com/dtd/web-app_2_3.dtd" >
5
6  <web-app>
7   <display-name>Archetype Created Web Application</display-name>
8   <servlet>
9   <servlet-name>springrest</servlet-name>
```

```
<servlet-class>
 org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>springrest</servlet-name>
 <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>

```

4) create a xml file named springrest-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to spring mvc hello world example for more understanding.**

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance"
 xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/sch
ema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.o
rg/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

 <mvc:annotation-driven/>
<context:component-scan base-package="org.arpit.java2blog.controller" />

</beans>
```

Create bean class

5) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```

package org.arpit.java2blog.bean;

import javax.xml.bind.annotation.XmlElement;
```

```
5  import javax.xml.bind.annotation.XmlRootElement;
6
7  @XmlRootElement(name="country")
8  public class Country{
9
10             private int id;
11             private String countryName;
12
13             public Country() {
14
15             }
16             public Country(int i, String countryName) {
17                         super();
18                         this.id = i;
19                         this.countryName = countryName;
20             }
21
22             @XmlElement
23             public int getId() {
24                         return id;
25             }
26             public void setId(int id) {
27                         this.id = id;
28             }
29
30             @XmlElement
31             public String getCountryName() {
32                         return countryName;
33             }
34             public void setCountryName(String countryName) {
35                         this.countryName = countryName;
36             }
37 }
38
```

We need to annotate bean class with @XmlRootElement and @XmlElement to support for xml. As you can see we have annotated Country class with

JAXB annotation but if you want to have support for list, we can not edit ArrayList class, so we can create another class called **CountryList** and we can annotate with JAXB annotation in that class to support xml output.

CountryList.java

```
1
2  package org.arpit.java2blog.bean;
3
4  import java.util.List;
5  import javax.xml.bind.annotation.XmlElement;
6  import javax.xml.bind.annotation.XmlRootElement;
7
8  @XmlRootElement(name = "country-list")
9  public class CountryList {
10
11              ListlistOfCountries;
12
13              public CountryList() {
14                      super();
15              }
16              public CountryList(ListlistOfCountries) {
17                      this.listOfCountries=listOfCountries;
18              }
19              public ListgetListOfCountries() {
20                      return listOfCountries;
21              }
22
23              @XmlElement(name = "country")
24              public void setListOfCountries(ListlistOfCountries) {
25                      this.listOfCountries = listOfCountries;
26              }
27
28 }
29
```

Create controller

6) Create a controller named “CountryController.java”

```
1
```

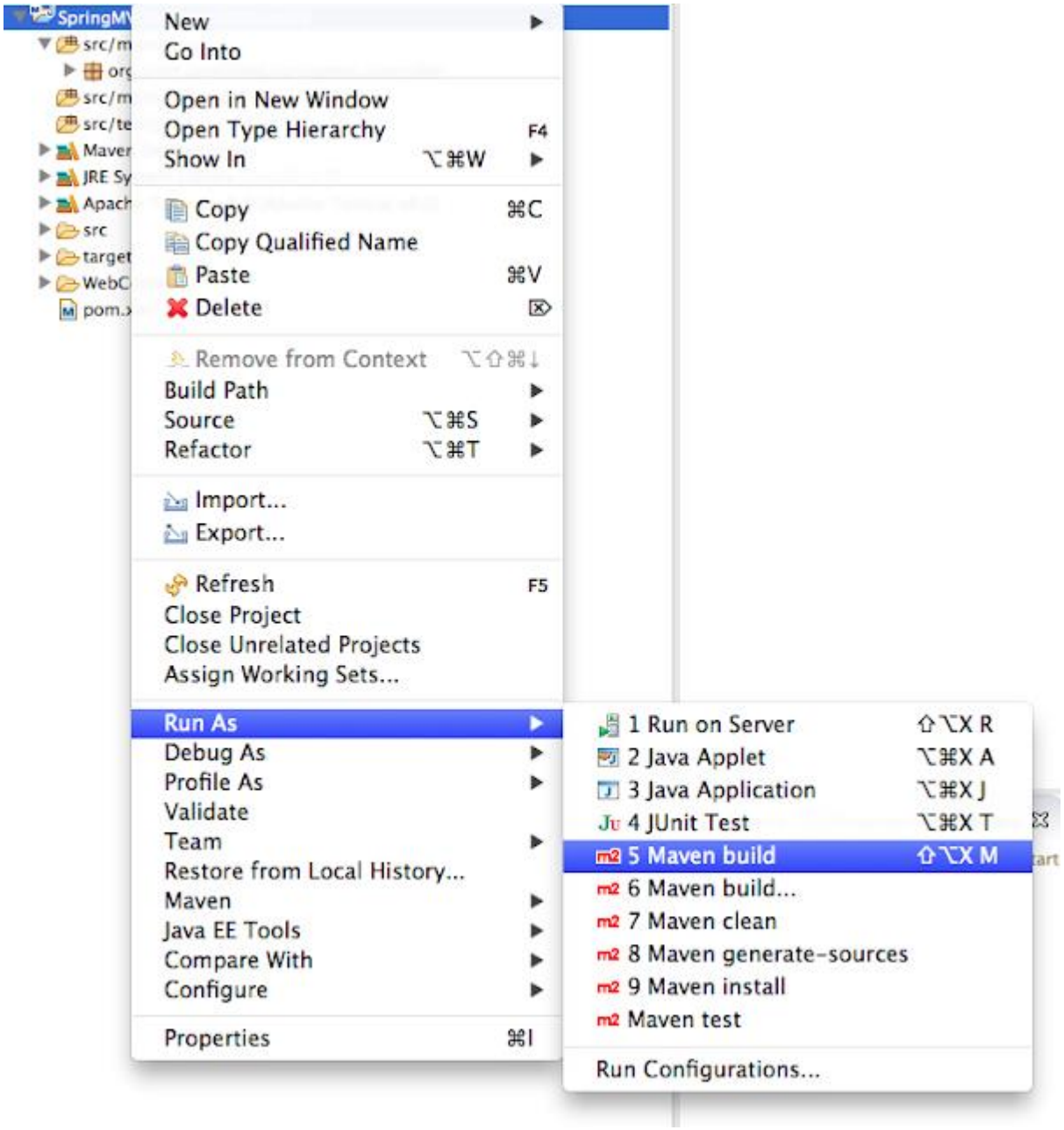
```
2  package org.arpit.java2blog.controller;
3
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import org.arpit.java2blog.bean.Country;
8  import org.arpit.java2blog.bean.CountryList;
9  import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RequestMethod;
12 import org.springframework.web.bind.annotation.RestController;
13
14 @RestController
15 public class CountryController {
16
17          @RequestMapping(value = "/countries", method = RequestMethod.GET,headers="Accept=application/
   xml")
18
19          public CountryList getCountries()
20          {
21                  CountryList countryList=createCountryList();
22                  return countryList;
23          }
24
25          @RequestMapping(value = "/country/{id}", method = RequestMethod.GET)
26          public Country getCountryById(@PathVariable int id)
27          {
28                  ListlistOfCountries = new ArrayList();
29                  CountryList countryList=createCountryList();
30                  listOfCountries=countryList.getListOfCountries();
31                  for (Country country: listOfCountries) {
32                          if(country.getId()==id)
33                                  return country;
34                  }
35
36                  return null;
            }
```

```
37
38          /// Utiliy method to create country list.
39          public CountryList createCountryList()
40          {
41                  Country indiaCountry=new Country(1, "India");
42                  Country chinaCountry=new Country(4, "China");
43                  Country nepalCountry=new Country(3, "Nepal");
44                  Country bhutanCountry=new Country(2, "Bhutan");
45
46                  ListlistOfCountries = new ArrayList();
47                  listOfCountries.add(indiaCountry);
48                  listOfCountries.add(chinaCountry);
49                  listOfCountries.add(nepalCountry);
50                  listOfCountries.add(bhutanCountry);
51                  return new CountryList(listOfCountries);
52          }
53 }
54
```

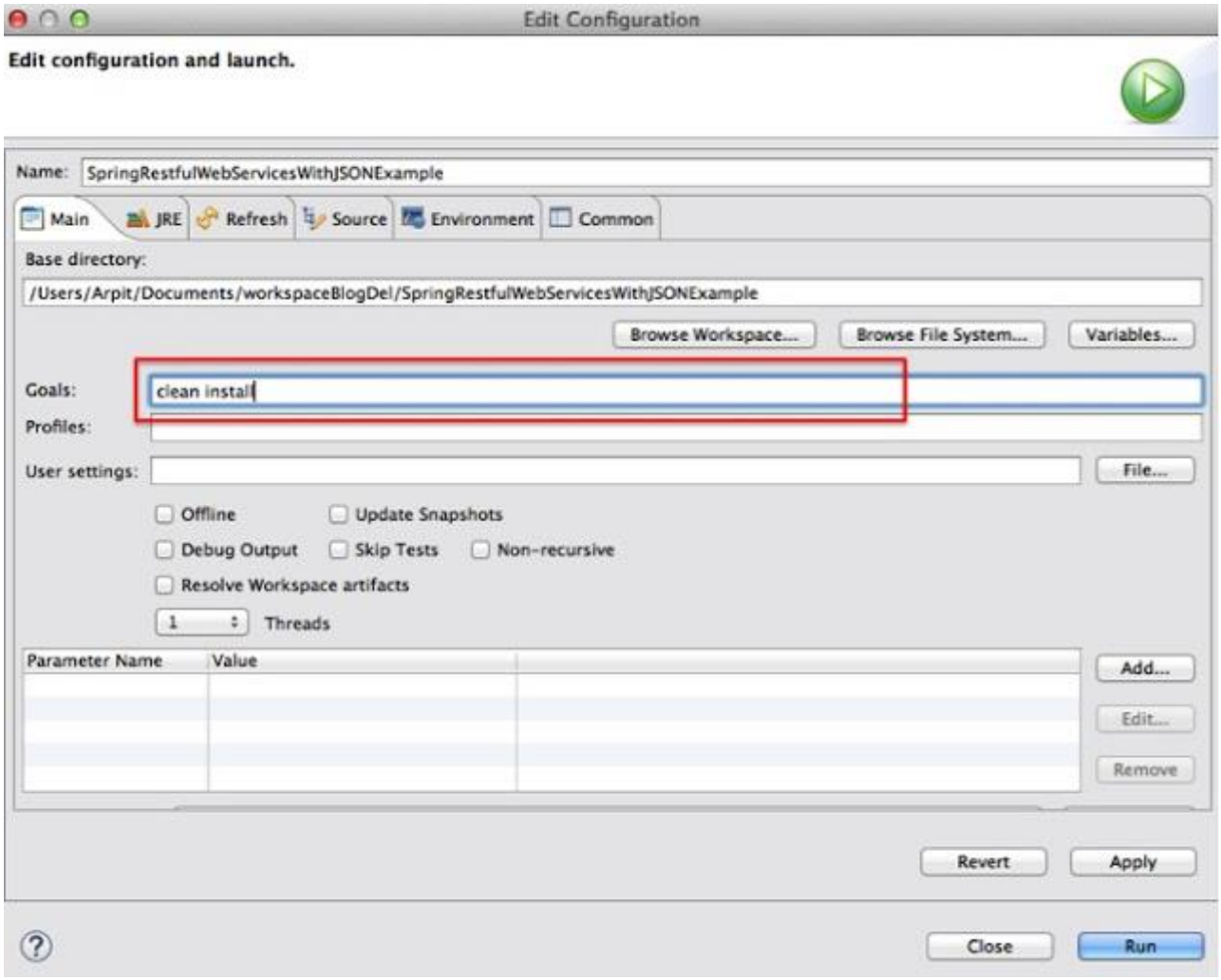
@PathVariable: Used to inject values from the URL into a method parameter.This way you inject id in getCountryById method .
We are not providing any view information in springrest-servlet.xml as we do in Spring MVC. If we need to directly get resource from controller, we need to return @ResponseBody as per Spring 3 but with Spring 4, we can use @RestController for that.
In spring 4.0, we can use @RestController which is combination of @Controller + @ResponseBody.

```
1
2 @RestController = @Controller + @ResponseBody
3
```

6) It 's time to do maven build.
Right click on project ->Run as ->Maven build



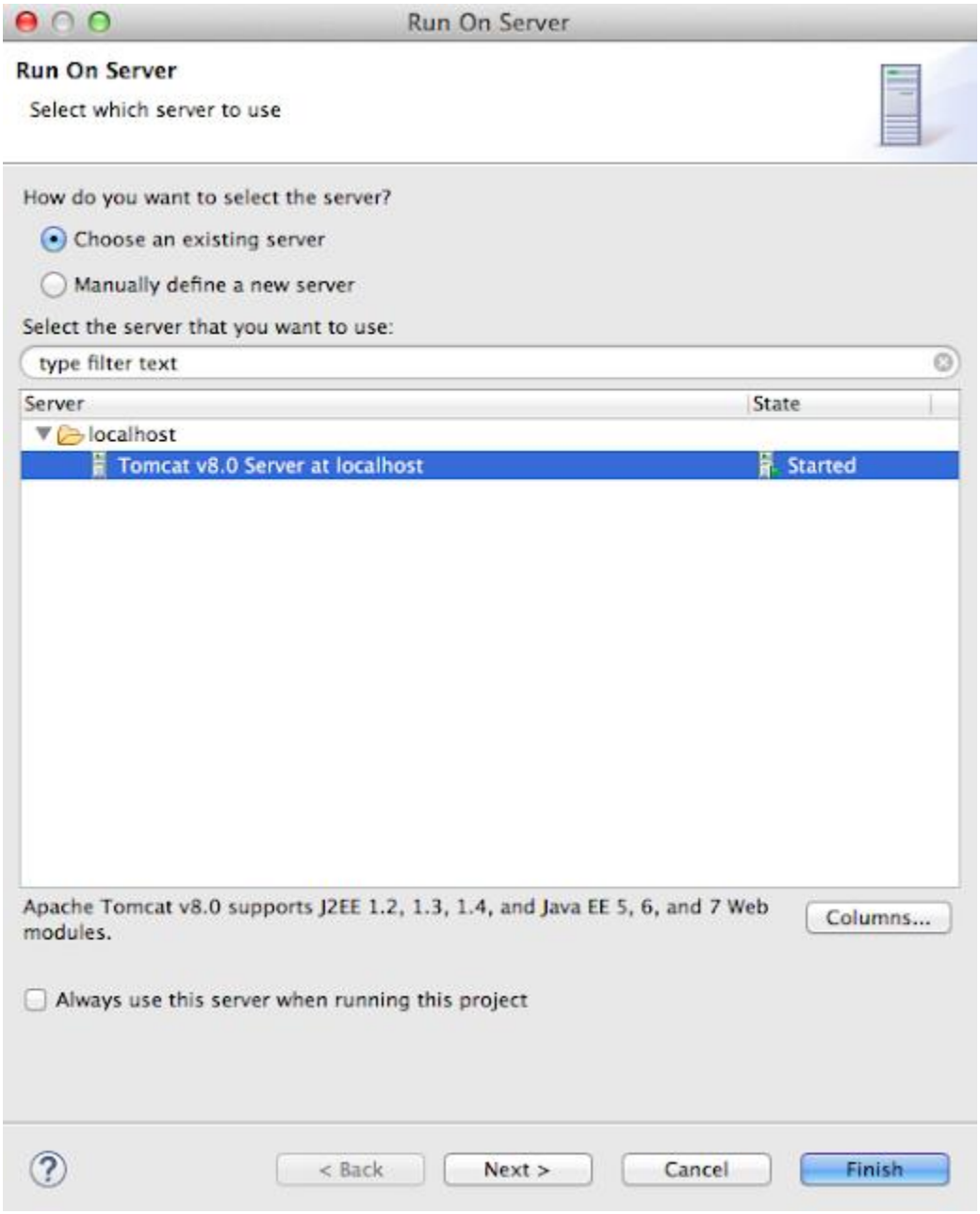
7) Provide goals as clean install (given below) and click on run



Run the application

8) Right click on project ->run as ->run on server

Select apache tomcat and click on finish



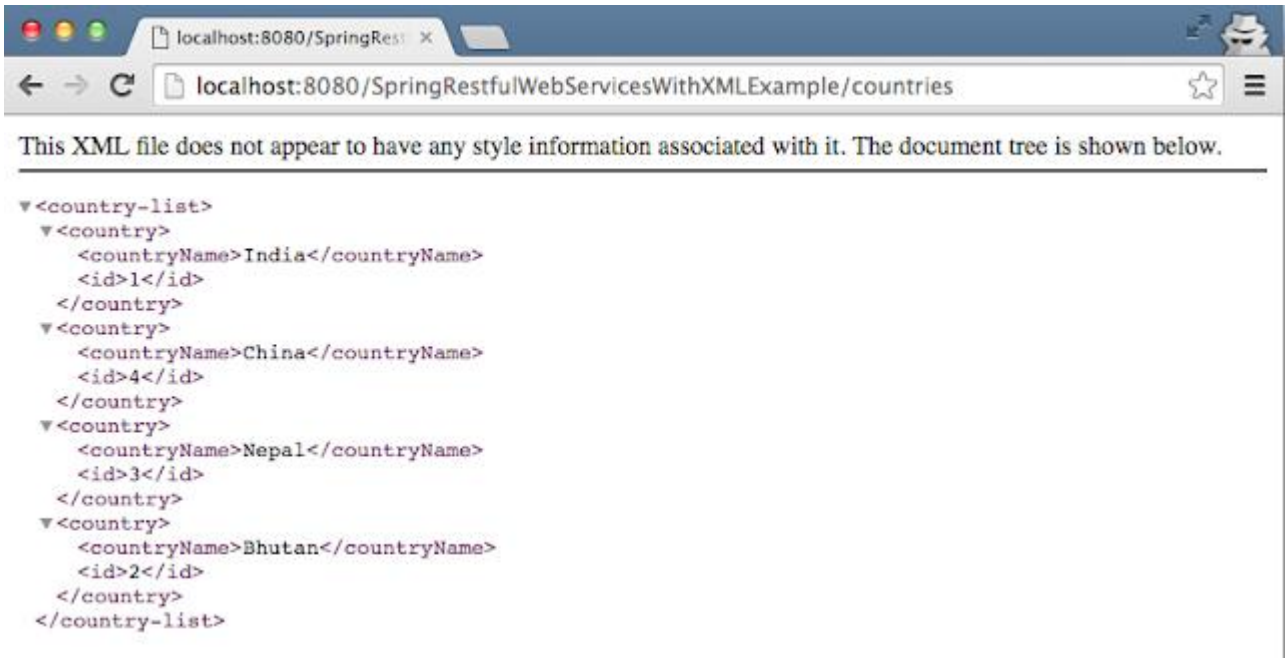
When you run the application, you might get this kind of warning

```
1
2 Mar 26, 2016 1:45:51 AM org.springframework.web.servlet.PageNotFound noHandlerFound
3 WARNING: No mapping found for HTTP request with URI [/SpringRestfulWebServicesWithXMLExample/] in Disp
  atcherServlet with name 'SpringRestfulWebServicesWithJSONExample'
4
```

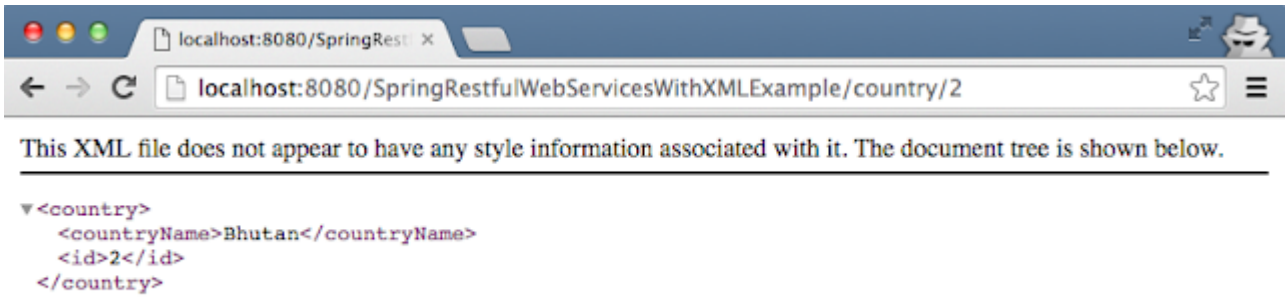
Please ignore above warning. When you start application, you have below URL if you have not provided start page: **http://localhost:8080/SpringRestfulWebServicesWithXMLExample/**

As we have used DispatcherServlet in web.xml, this request goes to spring DispatcherServlet and it did not find corresponding mapping in controller , hence you get that warning.

9) Test your REST service under: "http://localhost:8080/SpringRestfulWebServicesWithXMLExample/countries". You will get following output:



10) Now pass country id as a parameter to url. "http://localhost:8080/SpringRestfulWebServicesWithXMLExample/country/2".



Download

click to begin

20KB .zip

We are done with Spring Restful web services xml example.

Spring Restful web services CRUD example::

In previous post, we have already seen Spring Restful web services which returns json as response.In this post, we will extend same example and create Restful web services which will provide CRUD(Create, read, update and delete) operation example.If you want complete integration with hibernate and mysql, you can go through Spring Restful hibernate mysql example.
We will use following annotations for CRUD operation.

Method	Description
Get	It is used to read resource
Post	It is used to create new resource. It is not idempotent method
Put	It is generally used to update resource idempotent method
Delete	It is used to delete resource

Idempotent means result of multiple successful request will not change state of resource after initial application
For example :
Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.

Post is not idempotent method because when you use post to create resource , it will keep creating resource for each new request, so result of multiple successful request will not be same.

Source code:

Download source code:Spring Restful web services CRUD example

Here are steps to create a Spring Restful web services which will provide CRUD opertion.

1) Create a dynamic web project using maven in eclipse named "SpringRestfulWebServicesCRUDExample"

Maven dependencies

2) We need to add Jackson json utility in the classpath.

```
1
2 <dependency>
3         <groupId>com.fasterxml.jackson.core</groupId>
4         <artifactId>jackson-databind</artifactId>
5          <version>2.4.1</version>
6 </dependency>
7
```

Spring will load Jackson2JsonMessageConverter into its application context automatically. Whenever you request resource as json with accept headers=”Accept=application/json”, then Jackson2JsonMessageConverter comes into picture and convert resource to json format.

Now change pom.xml as follows:

pom.xml

```
1
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
3 nce"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6  <groupId>com.arpit.java2blog</groupId>
7  <artifactId>SpringRestfulWebServicesWithJSONExample</artifactId>
8  <packaging>war</packaging>
9  <version>0.0.1-SNAPSHOT</version>
10  <name>SpringRestfulWebServicesWithJSONExample Maven Webapp</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13   <dependency>
14    <groupId>junit</groupId>
15    <artifactId>junit</artifactId>
16    <version>3.8.1</version>
17    <scope>test</scope>
18   </dependency>
19   <dependency>
20  <groupId>javax.servlet</groupId>
```

```
21   <artifactId>javax.servlet-api</artifactId>
22   <version>3.1.0</version>
23  </dependency>
24
25  <dependency>
26   <groupId>org.springframework</groupId>
27   <artifactId>spring-core</artifactId>
28   <version>${spring.version}</version>
29  </dependency>
30  <dependency>
31   <groupId>org.springframework</groupId>
32   <artifactId>spring-webmvc</artifactId>
33   <version>${spring.version}</version>
34  </dependency>
35   <dependency>
36          <groupId>com.fasterxml.jackson.core</groupId>
37          <artifactId>jackson-databind</artifactId>
38           <version>2.4.1</version>
39       </dependency>
40 </dependencies>
41 <build>
42  <finalName>SpringRestfulWebServicesWithJSONExample</finalName>
43
44  <plugins>
45   <plugin>
46    <groupId>org.apache.maven.plugins</groupId>
47    <artifactId>maven-compiler-plugin</artifactId>
48    <version>3.1</version>
49    <configuration>
50     <source>${jdk.version}</source>
51     <target>${jdk.version}</target>
52    </configuration>
53   </plugin>
54  </plugins>
55
```

```
56  </build>
57  <properties>
58   <spring.version>4.2.1.RELEASE</spring.version>
59   <jdk.version>1.7</jdk.version>
60  </properties>
61 </project>
```

Spring application configuration:

3) Change web.xml as below:

```
1
2  <!DOCTYPE web-app PUBLIC
3   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4   "http://java.sun.com/dtd/web-app_2_3.dtd" >
5
6  <web-app>
7    <display-name>Archetype Created Web Application</display-name>
8    <servlet>
9   <servlet-name>springrest</servlet-name>
10  <servlet-class>
11   org.springframework.web.servlet.DispatcherServlet
12  </servlet-class>
13  <load-on-startup>1</load-on-startup>
14 </servlet>
15
16 <servlet-mapping>
17  <servlet-name>springrest</servlet-name>
18  <url-pattern>/</url-pattern>
19 </servlet-mapping>
20 </web-app>
21
```

4) create a xml file named springrest-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to spring mvc hello world example for more understanding.**

```
1
2  <beans xmlns="http://www.springframework.org/schema/beans"
```

```
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

 <mvc:annotation-driven/>
<context:component-scan base-package="org.arpit.java2blog.controller" />

</beans>
```

Create bean class

4) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```
package org.arpit.java2blog.bean;

public class Country{

	int id;
	String countryName;
	long population;

	public Country() {
		super();
	}
	public Country(int i, String countryName,long population) {
		super();
		this.id = i;
		this.countryName = countryName;
		this.population=population;
	}
	public int getId() {
		return id;
	}
```

```
22		public void setId(int id) {
23			this.id = id;
24		}
25		public String getCountryName() {
26			return countryName;
27		}
28		public void setCountryName(String countryName) {
29			this.countryName = countryName;
30		}
31		public long getPopulation() {
32			return population;
33		}
34		public void setPopulation(long population) {
35			this.population = population;
36		}
37
38 }
39
```

Create Controller

5) Create a controller named "CountryController.java" in package **org.arpit.java2blog.controller**

```
 1
 2 package org.arpit.java2blog.controller;
 3
 4 import java.util.List;
 5 import org.arpit.java2blog.bean.Country;
 6 import org.arpit.java2blog.service.CountryService;
 7 import org.springframework.web.bind.annotation.PathVariable;
 8 import org.springframework.web.bind.annotation.RequestBody;
 9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11 import org.springframework.web.bind.annotation.RestController;
12
13 @RestController
14 public class CountryController {
```

```
15
16              CountryService countryService = new CountryService();
17
18              @RequestMapping(value = "/countries", method = RequestMethod.GET, headers = "Accept=application
    /json")
19              public List getCountries() {
20                      List listOfCountries = countryService.getAllCountries();
21                      return listOfCountries;
22              }
23
24              @RequestMapping(value = "/country/{id}", method = RequestMethod.GET, headers = "Accept=applicat
25  ion/json")
26              public Country getCountryById(@PathVariable int id) {
27                      return countryService.getCountry(id);
28              }
29
30              @RequestMapping(value = "/countries", method = RequestMethod.POST, headers = "Accept=applicatio
31  n/json")
32              public Country addCountry(@RequestBody Country country) {
33                      return countryService.addCountry(country);
34              }
35
36              @RequestMapping(value = "/countries", method = RequestMethod.PUT, headers = "Accept=application
    /json")
37              public Country updateCountry(@RequestBody Country country) {
38                      return countryService.updateCountry(country);
39
40              }
41
42              @RequestMapping(value = "/country/{id}", method = RequestMethod.DELETE, headers = "Accept=app
43  lication/json")
44              public void deleteCountry(@PathVariable("id") int id) {
45                      countryService.deleteCountry(id);
46
                }
    }
```

Create Service class

6) Create a class CountryService.java in package **org.arpit.java2blog.service**

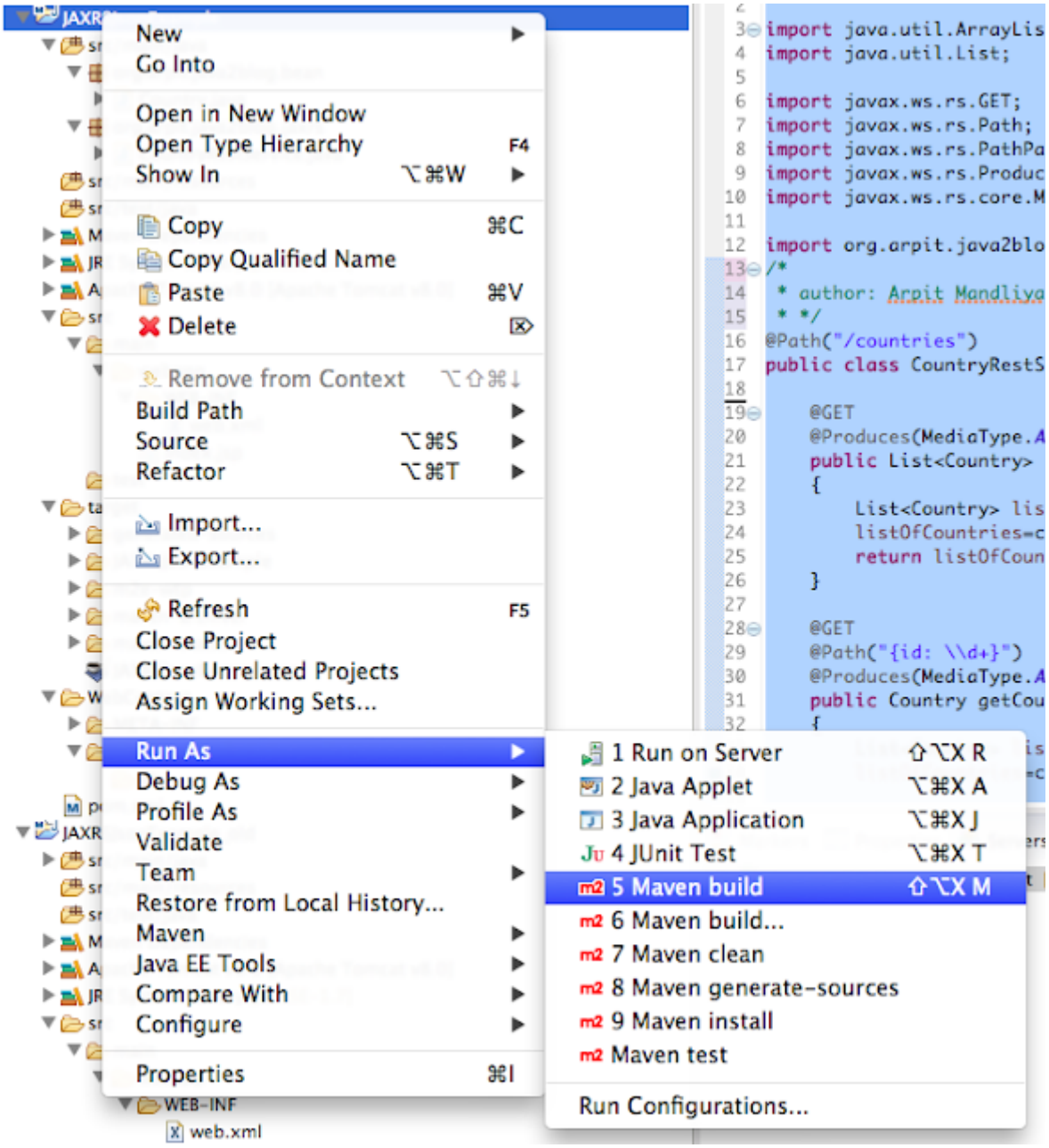
It is just a helper class which should be replaced by database implementation. It is not very well written class, it is just used for demonstration.

```
1
2  import java.util.ArrayList;
3  import java.util.HashMap;
4  import java.util.List;
5
6  import org.arpit.java2blog.bean.Country;
7
8  /*
9   * It is just a helper class which should be replaced by database implementation.
10  * It is not very well written class, it is just used for demonstration.
11  */
12 public class CountryService {
13
14          static HashMap<Integer,Country> countryIdMap=getCountryIdMap();
15
16
17          public CountryService() {
18                  super();
19
20                  if(countryIdMap==null)
21                  {
22                          countryIdMap=new HashMap<Integer,Country>();
23                          // Creating some objects of Country while initializing
24                          Country indiaCountry=new Country(1, "India",10000);
25                          Country chinaCountry=new Country(4, "China",20000);
26                          Country nepalCountry=new Country(3, "Nepal",8000);
27                          Country bhutanCountry=new Country(2, "Bhutan",7000);
28
29
30                          countryIdMap.put(1,indiaCountry);
```

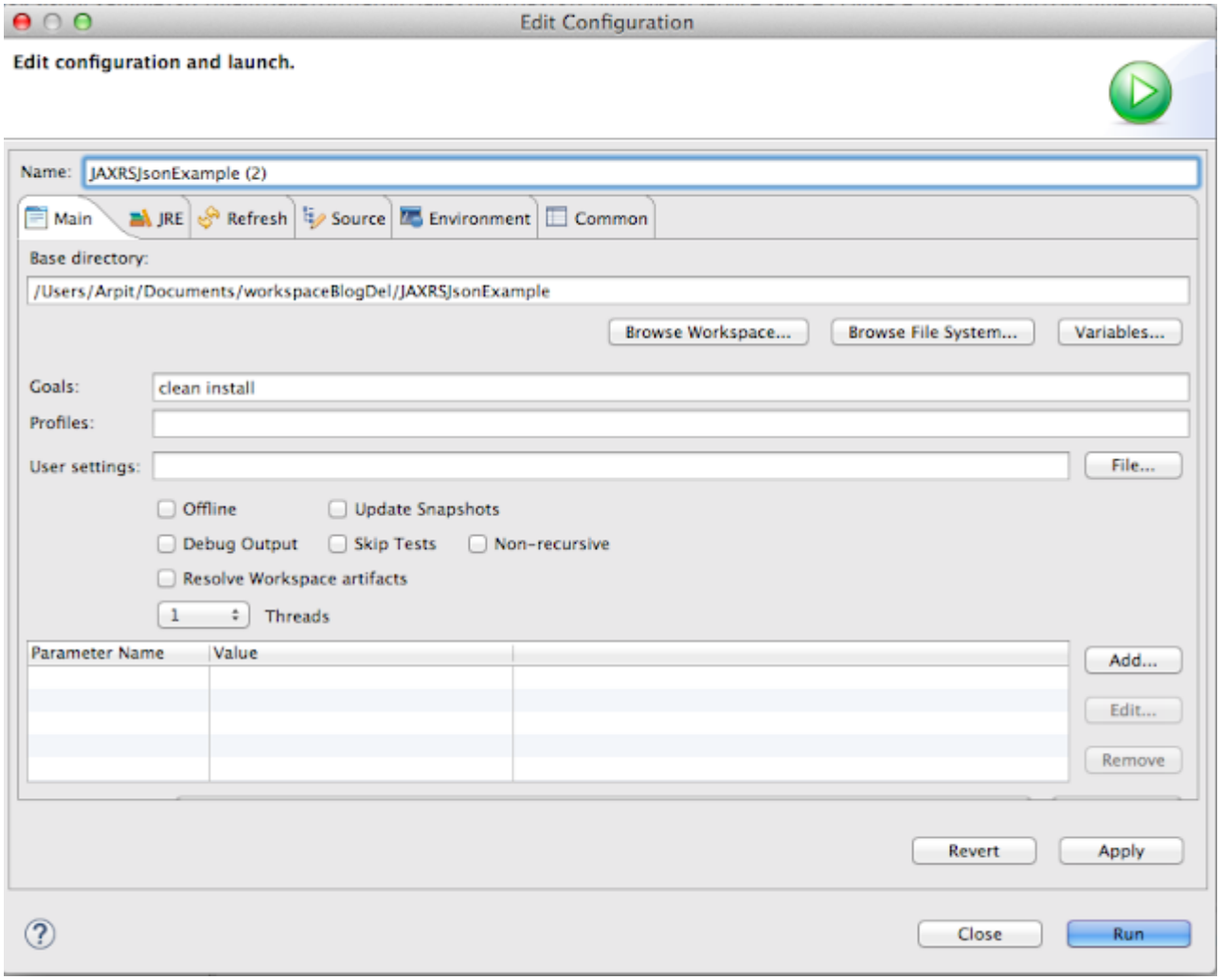
```
31                              countryIdMap.put(4,chinaCountry);
32                              countryIdMap.put(3,nepalCountry);
33                              countryIdMap.put(2,bhutanCountry);
34                      }
35              }
36
37              public List getAllCountries()
38              {
39                      List countries = new ArrayList(countryIdMap.values());
40                      return countries;
41              }
42
43              public Country getCountry(int id)
44              {
45                      Country country= countryIdMap.get(id);
46                      return country;
47              }
48              public Country addCountry(Country country)
49              {
50                      country.setId(getMaxId()+1);
51                      countryIdMap.put(country.getId(), country);
52                      return country;
53              }
54
55              public Country updateCountry(Country country)
56              {
57                      if(country.getId()<=0)
58                              return null;
59                      countryIdMap.put(country.getId(), country);
60                      return country;
61
62              }
63              public void deleteCountry(int id)
64              {
65                      countryIdMap.remove(id);
```

```
66              }
67
68              public static HashMap<Integer, Country> getCountryIdMap() {
69                              return countryIdMap;
70              }
71
72              // Utility method to get max id
73              public static int getMaxId()
74              { int max=0;
75              for (int id:countryIdMap.keySet()) {
76                              if(max<=id)
77                                              max=id;
78
79              }
80              return max;
81              }
82 }
83
```

7) It ‘s time to do maven build.
Right click on project -> Run as -> Maven build



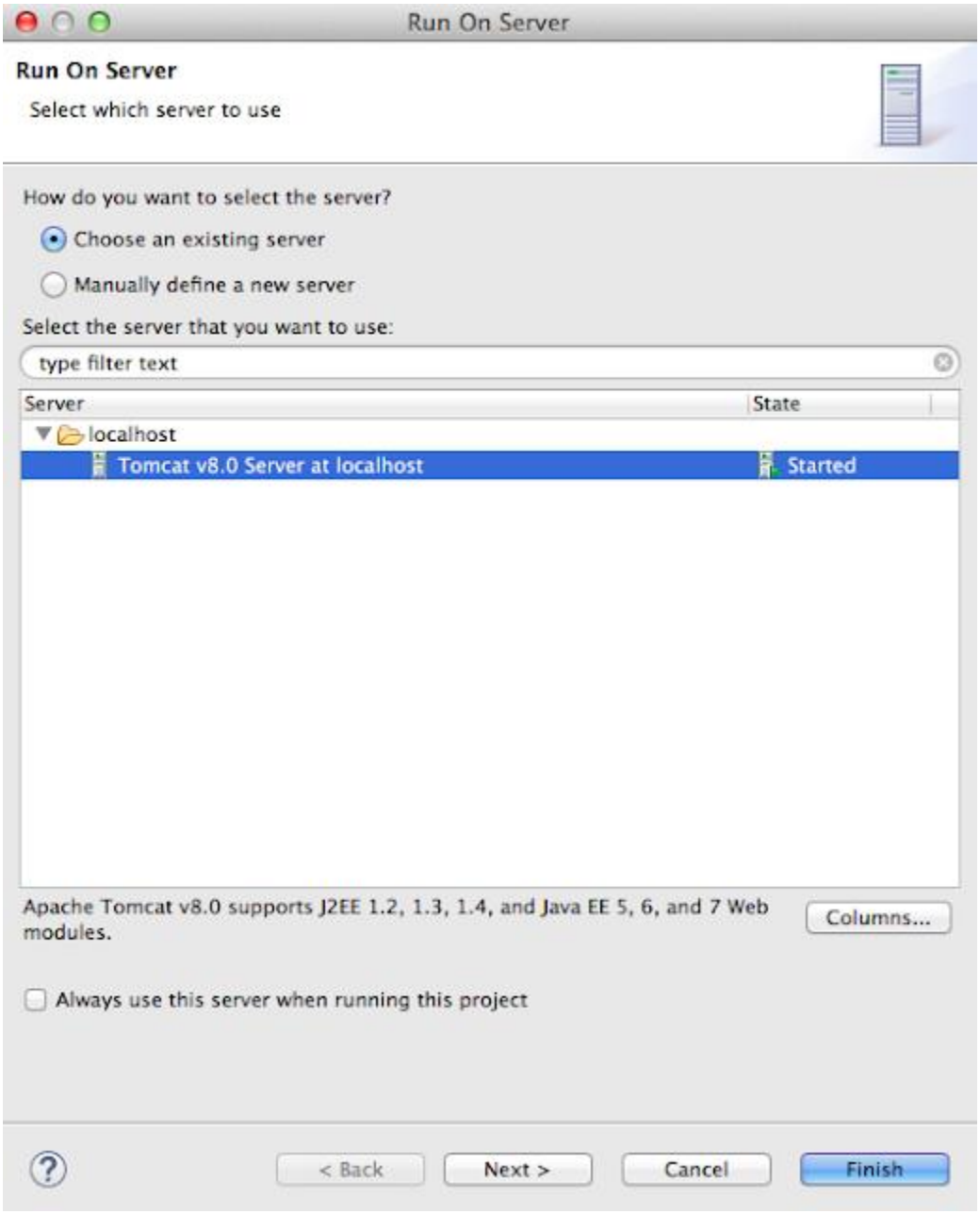
8) Provide goals as clean install (given below) and click on run



Run the application

9) Right click on project -> run as -> run on server

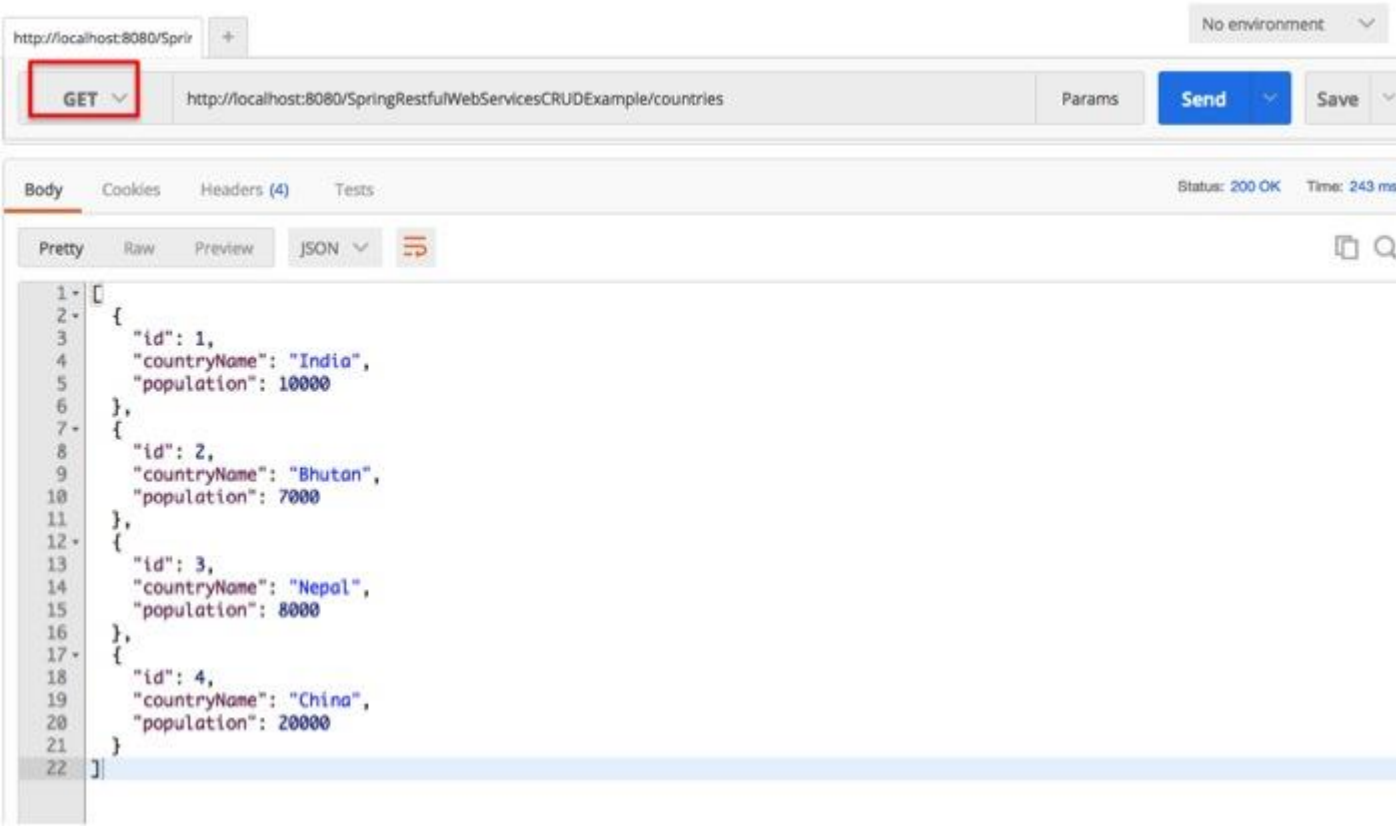
Select apache tomcat and click on finish



10) We will test this application in postman , UI based client for testing restful web applications. It is chrome plugin. Launch postman.If you want java based client, then you can also use how to send get or post request in java.

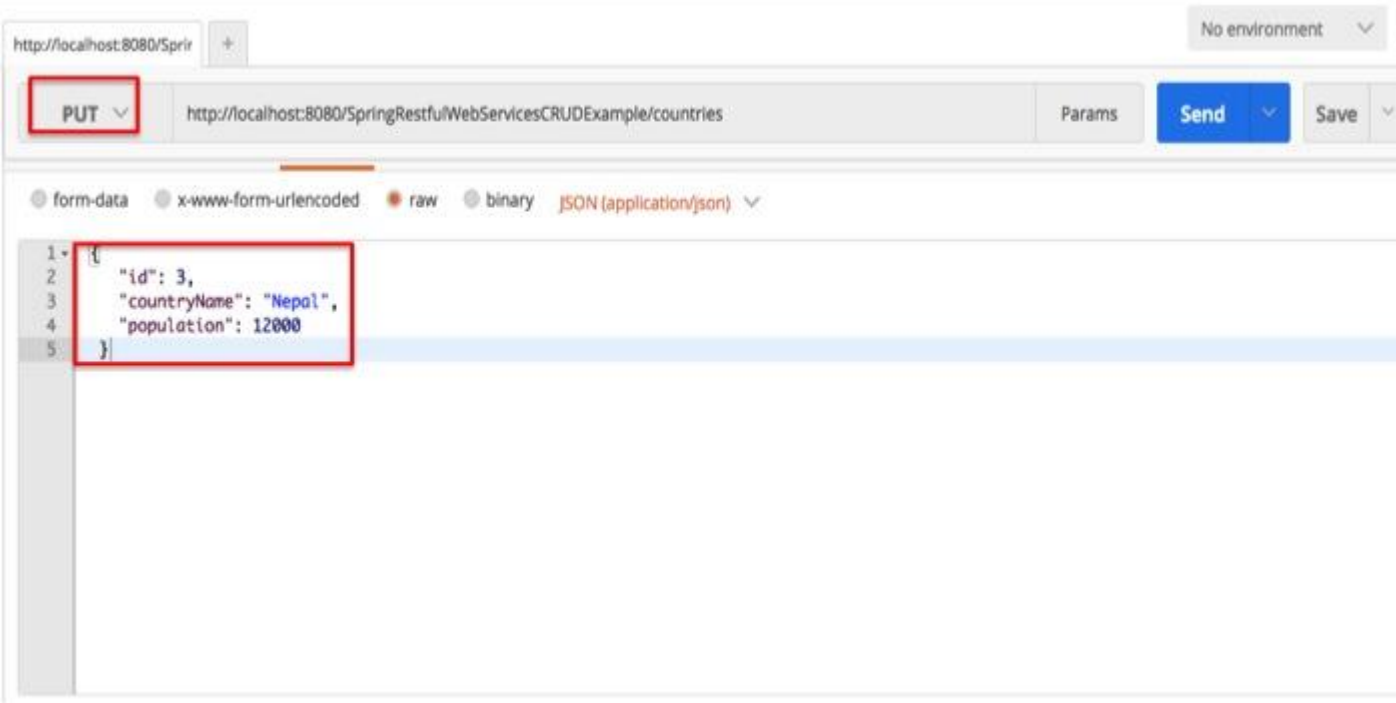
Get method

11) Test your get method Spring REST service **URL :**"http://localhost:8080/SpringRestfulWebServicesCRUDExample/countries". You will get following output:

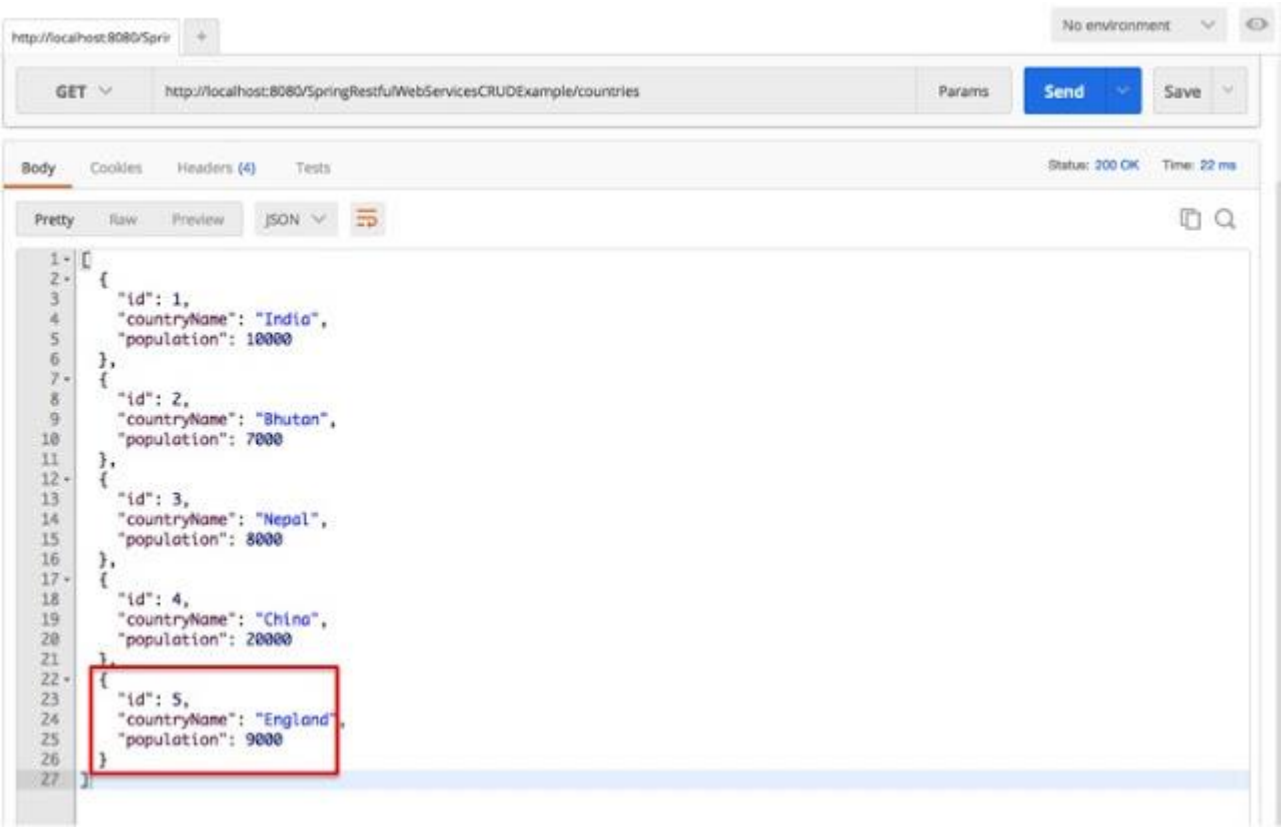


Post method

12) Post method is used to create new resource. Here we are adding new Country England to country list, so you can see we have used new country json in post body.
URL: "http://localhost:8080/SpringRestfulWebServicesCRUDExample/countries".

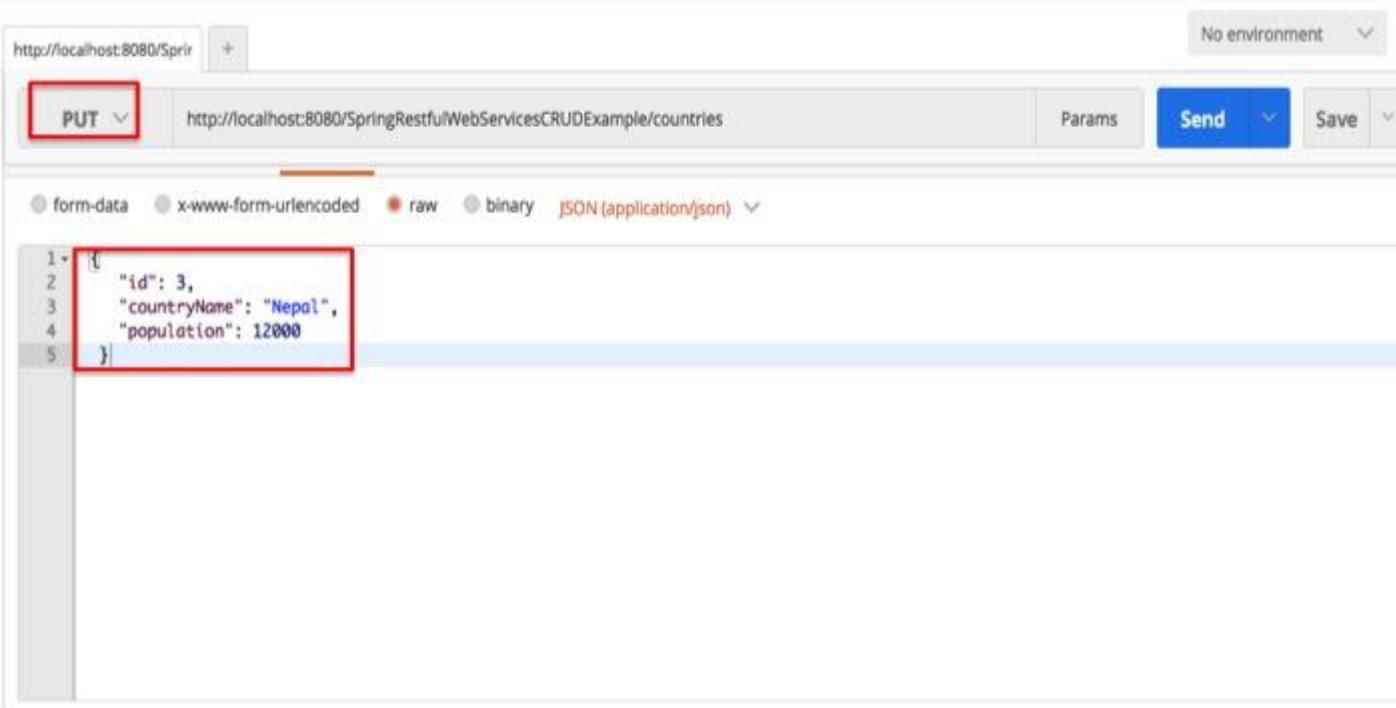


Use get method to check if above country have been added to country list.

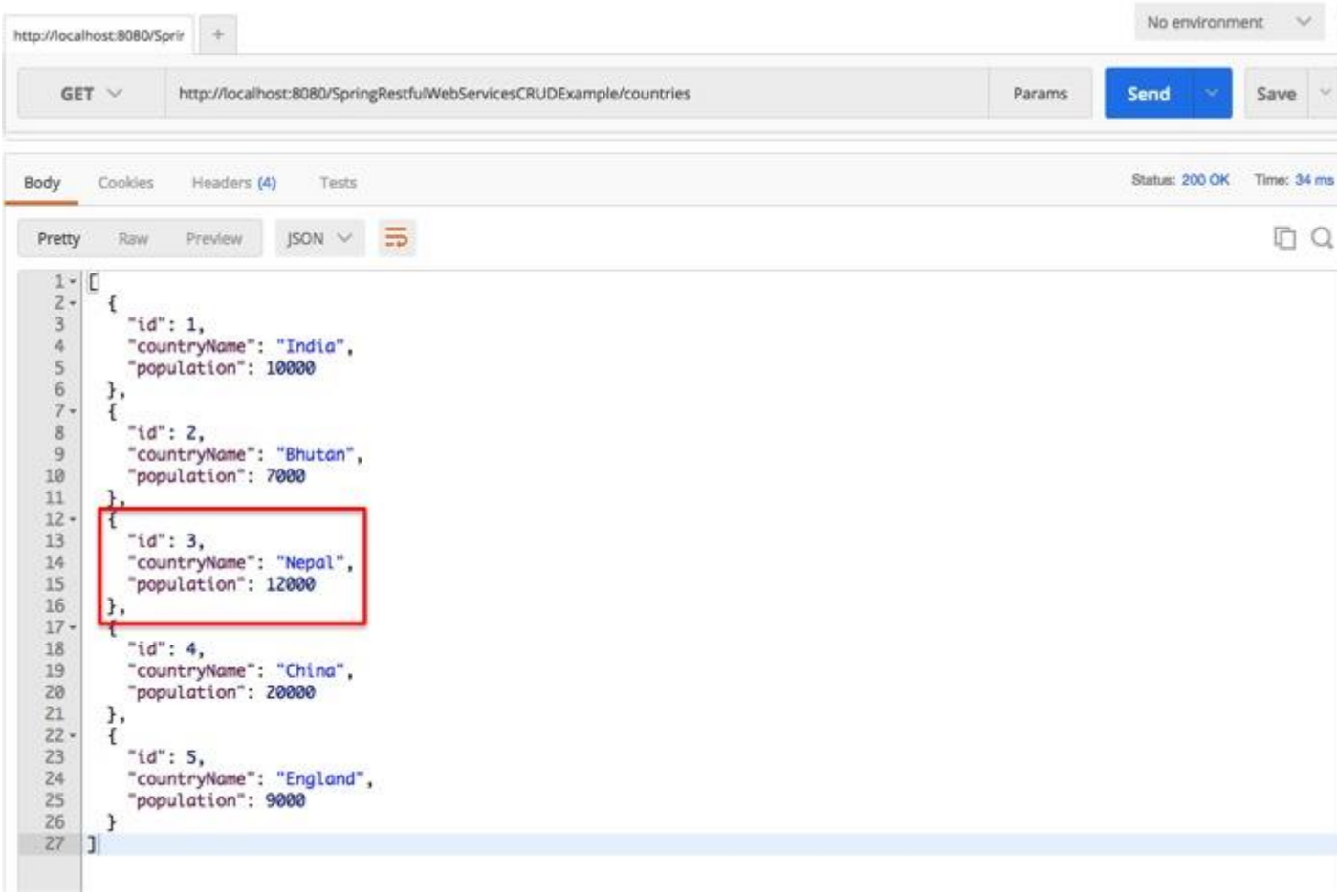


Put Method

13) Put method is used to update resource. Here will update population of nepal using put method. We will update country json in body of request.
URL : "http://localhost:8080/SpringRestfulWebServicesCRUDExample/countries"



Use get method to check population of nepal.

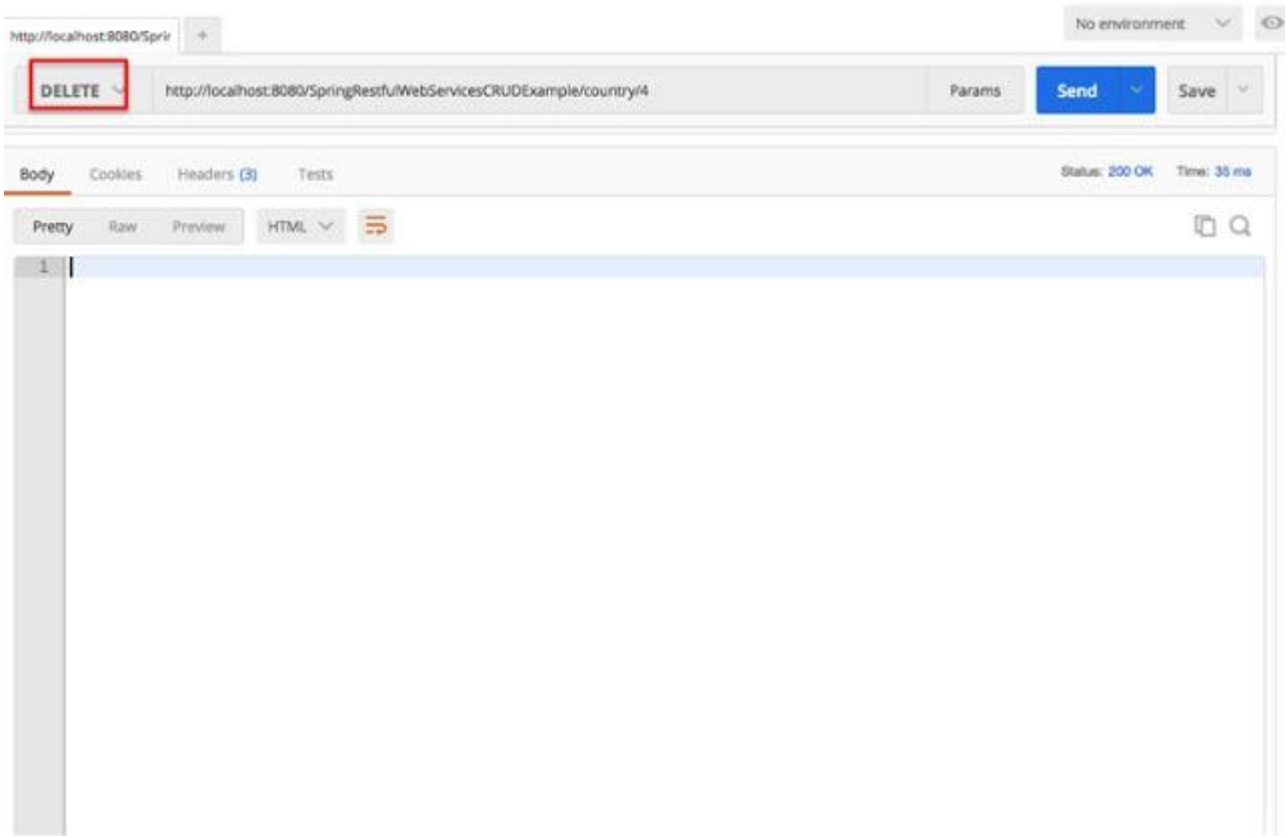


Delete method

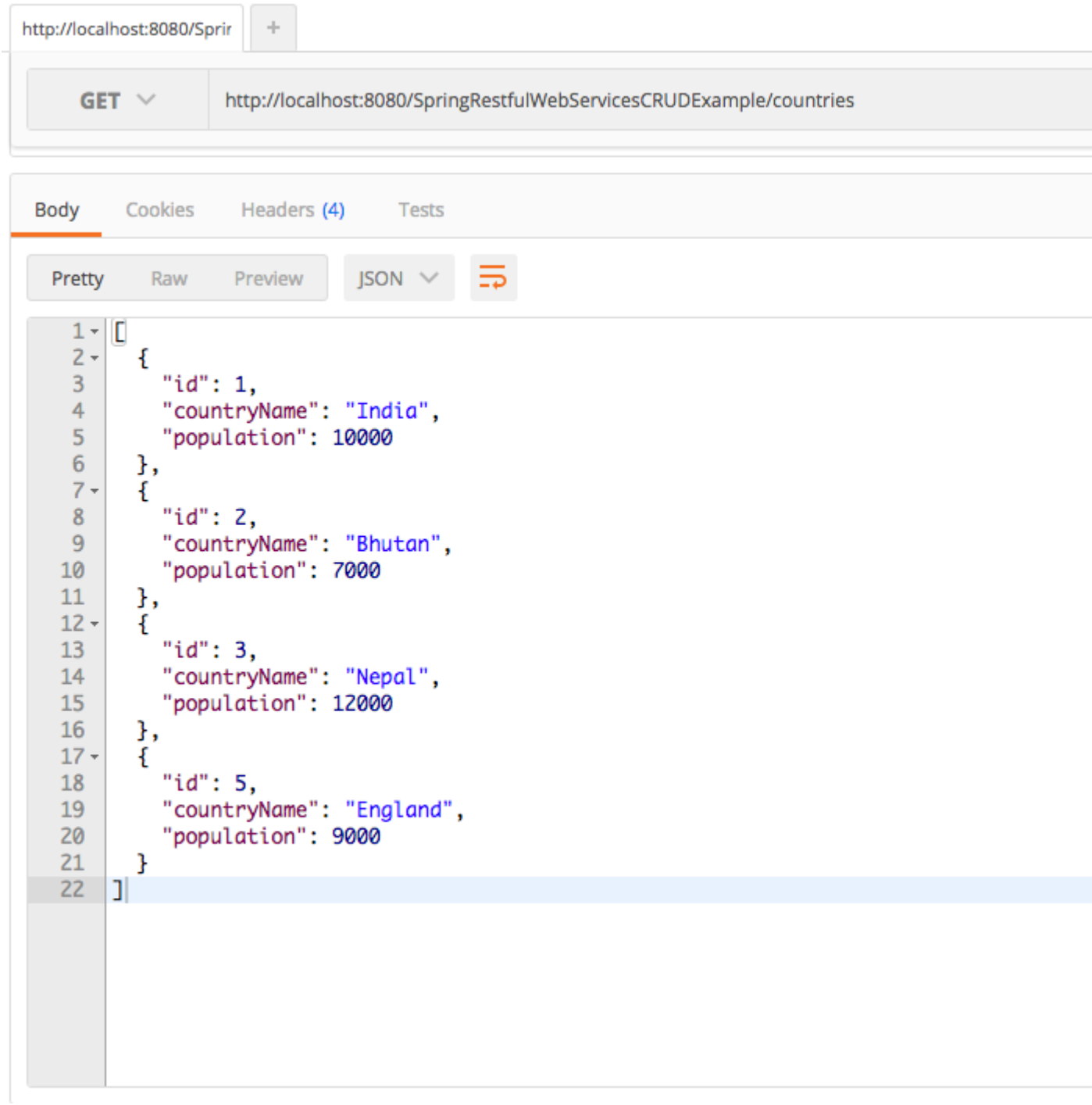
14) Delete method is used to delete resource.We will pass id of country which needs to be deleted as PathParam. We are going delete id:4 i.e. china to demonstrate delete method.

URL

: "http://localhost:8080/SpringRestfulWebServicesCRUDExample/country/4"

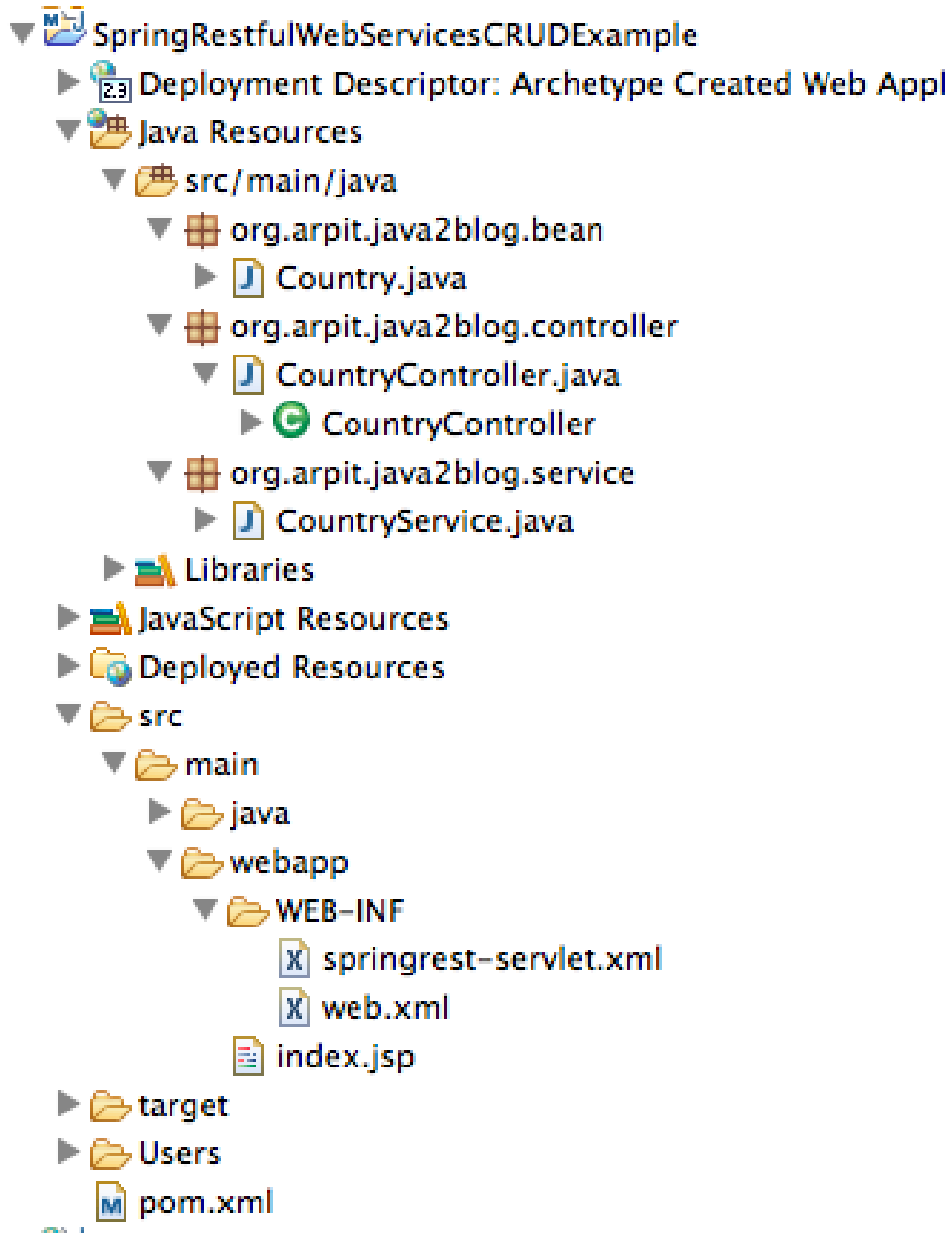


Use get method to check country list.



As you can see, we have deleted **country with id 4 i.e. china**

Project structure:



We are done with Spring Restful web services json CRUD example.

Spring Rest Hibernate example::

In previous post, we have already seen Spring Rest crud example . In this post, we will extend same example and integrate it with **hibernate** and **mysql**.
We will use following annotations for CRUD operation.

Method	Description

Get	It is used to read resource
Post	It is used to create new resource. It is not idempotent method
Put	It is generally used to update resource.It is idempo method
Delete	It is used to delete resource

Idempotent means result of multiple successful request will not change state of resource after initial application
For example :
Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.
Post is not idempotent method because when you use post to create resource , it will keep creating resource for each new request, so result of multiple successful request will not be same.

Source code:

Download Spring rest hibernate example

Here are steps to create a Spring Restful web services with hibernate integration.

1) Create a dynamic web project using maven in eclipse named "SpringRestHibernateExample"

Maven dependencies

2) We need to add Jackson json utility in the classpath.

```
1
2 <dependency>
3         <groupId>com.fasterxml.jackson.core</groupId>
4         <artifactId>jackson-databind</artifactId>
5          <version>2.4.1</version>
6 </dependency>
7
```

Spring will load Jackson2JsonMessageConverter into its application context automatically. Whenever you request resource as json with accept headers="Accept=application/json", then Jackson2JsonMessageConverter comes into picture and convert resource to json format.
Now change pom.xml as follows:

pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
3  tance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.arpit.java2blog</groupId>
<artifactId>SpringRestHibernateExample</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringRestHibernateExample Maven Webapp</name>
<url>http://maven.apache.org</url>
<dependencies>
 <dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
 </dependency>
 <dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
 </dependency>

 <dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>${spring.version}</version>
 </dependency>
 <dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
 </dependency>
 <dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.4.1</version>
 </dependency>
 <!-- Hibernate -->
 <dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>${hibernate.version}</version>
 </dependency>
 <dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>${hibernate.version}</version>
 </dependency>

 <!-- Apache Commons DBCP -->
 <dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
 </dependency>
 <!-- Spring ORM -->
 <dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
 </dependency>

 <!-- AspectJ -->
 <dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
```

```
    <version>${org.aspectj-version}</version>
   </dependency>
   <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
   </dependency>
  </dependencies>
  <build>
   <finalName>SpringRestHibernateExample</finalName>

   <plugins>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.1</version>
     <configuration>
      <source>${jdk.version}</source>
      <target>${jdk.version}</target>
     </configuration>
    </plugin>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-war-plugin</artifactId>
     <configuration>
      <failOnMissingWebXml>false</failOnMissingWebXml>
     </configuration>
    </plugin>
   </plugins>

  </build>
  <properties>
   <spring.version>4.2.1.RELEASE</spring.version>
   <security.version>4.0.3.RELEASE</security.version>
   <jdk.version>1.7</jdk.version>
   <hibernate.version>4.3.5.Final</hibernate.version>
   <org.aspectj-version>1.7.4</org.aspectj-version>
  </properties>
 </project>
```

4) create a xml file named spring-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to [spring mvc hello world example] for more understanding.**

```

 <?xml version="1.0" encoding="UTF-8"?>
 <beans:beans xmlns="http://www.springframework.org/schema/mvc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.springframework.org/
 schema/beans"
  xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.
 org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/sche
 ma/mvc/spring-mvc.xsd
  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-be
 ans.xsd
  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring
 -context.xsd
  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
 ">

 <annotation-driven />

 <resources mapping="/resources/**" location="/resources/" />
```

```
20
21  <beans:bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
22   destroy-method="close">
23   <beans:property name="driverClassName" value="com.mysql.jdbc.Driver" />
24   <beans:property name="url"
25    value="jdbc:mysql://localhost:3306/CountryData" />
26   <beans:property name="username" value="root" />
27   <beans:property name="password" value="" />
28  </beans:bean>
29
30  <!-- Hibernate 4 SessionFactory Bean definition -->
31  <beans:bean id="hibernate4AnnotatedSessionFactory"
32   class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
33   <beans:property name="dataSource" ref="dataSource" />
34   <beans:property name="annotatedClasses">
35    <beans:list>
36     <beans:value>org.arpit.java2blog.model.Country</beans:value>
37    </beans:list>
38   </beans:property>
39   <beans:property name="hibernateProperties">
40    <beans:props>
41     <beans:prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect
42     </beans:prop>
43     <beans:prop key="hibernate.show_sql">true</beans:prop>
44    </beans:props>
45   </beans:property>
46  </beans:bean>
47
48  <context:component-scan base-package="org.arpit.java2blog" />
49
50  <tx:annotation-driven transaction-manager="transactionManager"/>
51
52  <beans:bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionMana
    ger">
     <beans:property name="sessionFactory" ref="hibernate4AnnotatedSessionFactory" />
     </beans:bean>


    </beans:beans>
```

In Spring-servlet.xml, we have done hibernate configuration. **dataSource** bean is used to specify java data source. We need to provide driver, URL , Username and Password. **transactionManager** bean is used to configure hibernate transaction manager. **hibernate4AnnotatedSessionFactory** bean is used to configure FactoryBean that creates a Hibernate SessionFactory. This is the common way to set up a shared Hibernate SessionFactory in a Spring application context, so you can use this SessionFactory to inject in Hibernate data access objects.

Create bean class

4) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```
1
2  package org.arpit.java2blog.model;
3
4  import javax.persistence.Column;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9  import javax.persistence.Table;
```

```
10
11 /*
12  * This is our model class and it corresponds to Country table in database
13  */
14 @Entity
15 @Table(name="COUNTRY")
16 public class Country{
17
18              @Id
19              @Column(name="id")
20              @GeneratedValue(strategy=GenerationType.IDENTITY)
21              int id;
22
23              @Column(name="countryName")
24              String countryName;
25
26              @Column(name="population")
27              long population;
28
29              public Country() {
30                      super();
31              }
32              public Country(int i, String countryName,long population) {
33                      super();
34                      this.id = i;
35                      this.countryName = countryName;
36                      this.population=population;
37              }
38              public int getId() {
39                      return id;
40              }
41              public void setId(int id) {
42                      this.id = id;
43              }
44              public String getCountryName() {
45                      return countryName;
46              }
47              public void setCountryName(String countryName) {
48                      this.countryName = countryName;
49              }
50              public long getPopulation() {
51                      return population;
52              }
53              public void setPopulation(long population) {
54                      this.population = population;
55              }
56
57 }
58
```

@Entity is used for making a persistent pojo class.For this java class,you will have corresponding table in database. @Column is used to map annotated attribute to corresponding column in table. So Create Country table in mysql database with following code:

```
1
2 CREATE TABLE COUNTRY
3 (
4   id int PRIMARY KEY NOT NULL AUTO_INCREMENT,
5   countryName varchar(100) NOT NULL,
6   population int NOT NULL
7 )
8 ;
9
```

Create Controller

5) Create a controller named “CountryController.java” in package **org.arpit.java2blog.controller**

```
package org.arpit.java2blog.controller;

import java.util.List;
import org.arpit.java2blog.model.Country;
import org.arpit.java2blog.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CountryController {

	@Autowired
	CountryService countryService;

	@RequestMapping(value = "/getAllCountries", method = RequestMethod.GET, headers = "Accept=appli
cation/json")
	public List getCountries() {

		List listOfCountries = countryService.getAllCountries();
		return listOfCountries;
	}

	@RequestMapping(value = "/getCountry/{id}", method = RequestMethod.GET, headers = "Accept=appli
cation/json")
	public Country getCountryById(@PathVariable int id) {
		return countryService.getCountry(id);
	}

	@RequestMapping(value = "/addCountry", method = RequestMethod.POST, headers = "Accept=applicat
ion/json")
	public void addCountry(@RequestBody Country country) {
		countryService.addCountry(country);

	}

	@RequestMapping(value = "/updateCountry", method = RequestMethod.PUT, headers = "Accept=appli
cation/json")
	public void updateCountry(@RequestBody Country country) {
		countryService.updateCountry(country);
	}

	@RequestMapping(value = "/deleteCountry/{id}", method = RequestMethod.DELETE, headers = "Accep
t=application/json")
	public void deleteCountry(@PathVariable("id") int id) {
		countryService.deleteCountry(id);
	}
}
```

Create DAO class

Create a class **CountryDAO.java** in package **org.arpit.java2blog.dao.** This class will execute hibernate statements while interacting with database.

```

package org.arpit.java2blog.dao;

```

```
import java.util.List;

import org.arpit.java2blog.model.Country;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class CountryDAO {

	@Autowired
	private SessionFactory sessionFactory;

	public void setSessionFactory(SessionFactory sf) {
		this.sessionFactory = sf;
	}

	public List getAllCountries() {
		Session session = this.sessionFactory.getCurrentSession();
		List countryList = session.createQuery("from Country").list();
		return countryList;
	}

	public Country getCountry(int id) {
		Session session = this.sessionFactory.getCurrentSession();
		Country country = (Country) session.load(Country.class, new Integer(id));
		return country;
	}

	public Country addCountry(Country country) {
		Session session = this.sessionFactory.getCurrentSession();
		session.persist(country);
		return country;
	}

	public void updateCountry(Country country) {
		Session session = this.sessionFactory.getCurrentSession();
		session.update(country);
	}

	public void deleteCountry(int id) {
		Session session = this.sessionFactory.getCurrentSession();
		Country p = (Country) session.load(Country.class, new Integer(id));
		if (null != p) {
			session.delete(p);
		}
	}
}

```

@Repository is specialised component annotation which is used to create bean at DAO layer. We have use Autowired annotation to inject hibernate SessionFactory into CountryDAO class. We have already configured hibernate SessionFactory object in Spring-Servlet.xml file.

Create Service class

6) Create a class CountryService.java in package **org.arpit.java2blog.service**

It is service level class. It will call DAO layer class.

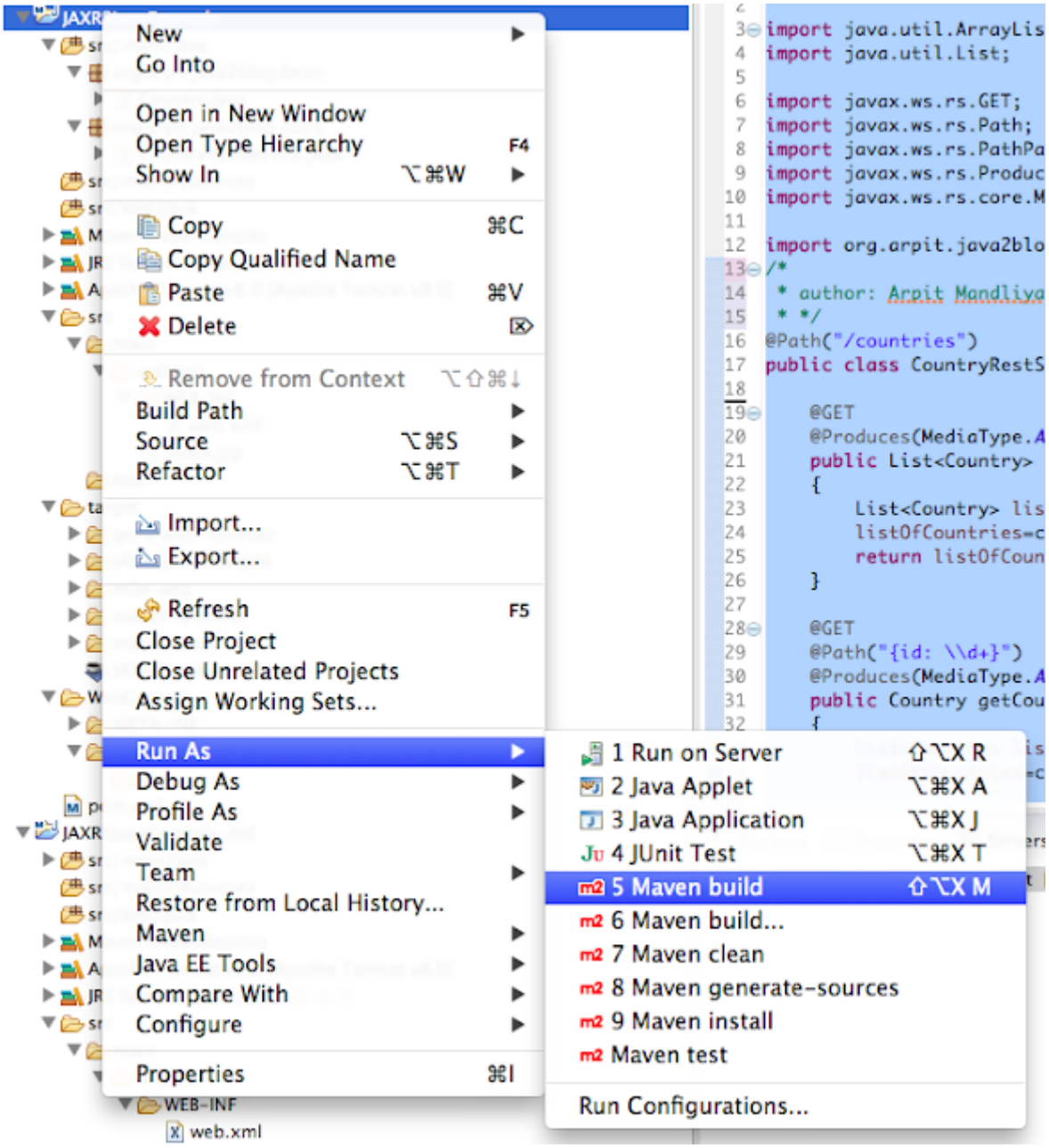
```

package org.arpit.java2blog.service;

```

```
 4 import java.util.List;
 5 import org.arpit.java2blog.dao.CountryDAO;
 6 import org.arpit.java2blog.model.Country;
 7 import org.springframework.beans.factory.annotation.Autowired;
 8 import org.springframework.stereotype.Service;
 9 import org.springframework.transaction.annotation.Transactional;
10
11 @Service("countryService")
12 public class CountryService {
13
14 	@Autowired
15 	CountryDAO countryDao;
16
17 	@Transactional
18 	public List getAllCountries() {
19 		return countryDao.getAllCountries();
20 	}
21
22 	@Transactional
23 	public Country getCountry(int id) {
24 		return countryDao.getCountry(id);
25 	}
26
27 	@Transactional
28 	public void addCountry(Country country) {
29 		countryDao.addCountry(country);
30 	}
31
32 	@Transactional
33 	public void updateCountry(Country country) {
34 		countryDao.updateCountry(country);
35
36 	}
37
38 	@Transactional
39 	public void deleteCountry(int id) {
40 		countryDao.deleteCountry(id);
41 	}
42 }
43
```

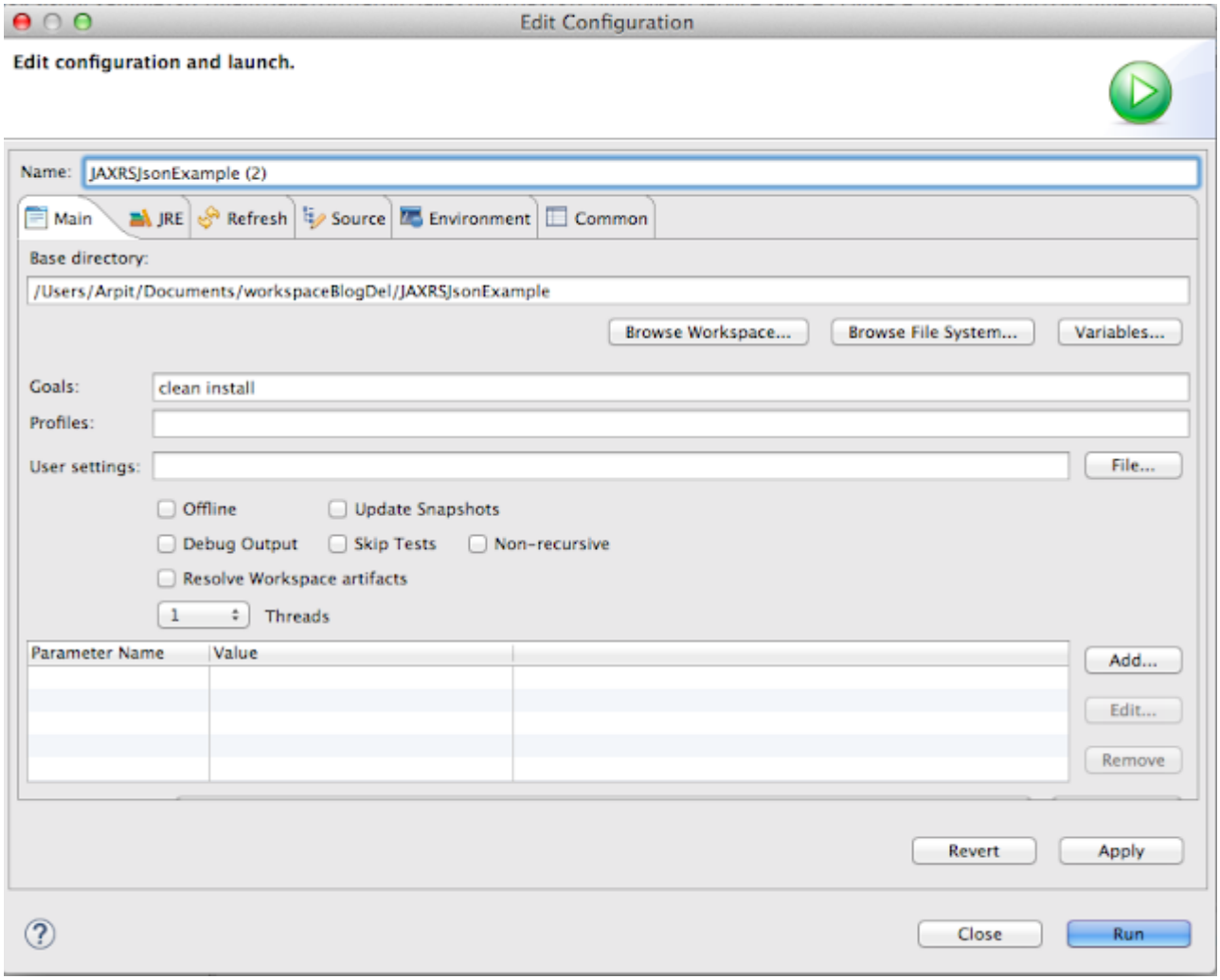
@Service is specialised component annotation which is used to create bean at Service layer.

7) It 's time to do maven build.

Right click on project -> Run as -> Maven build

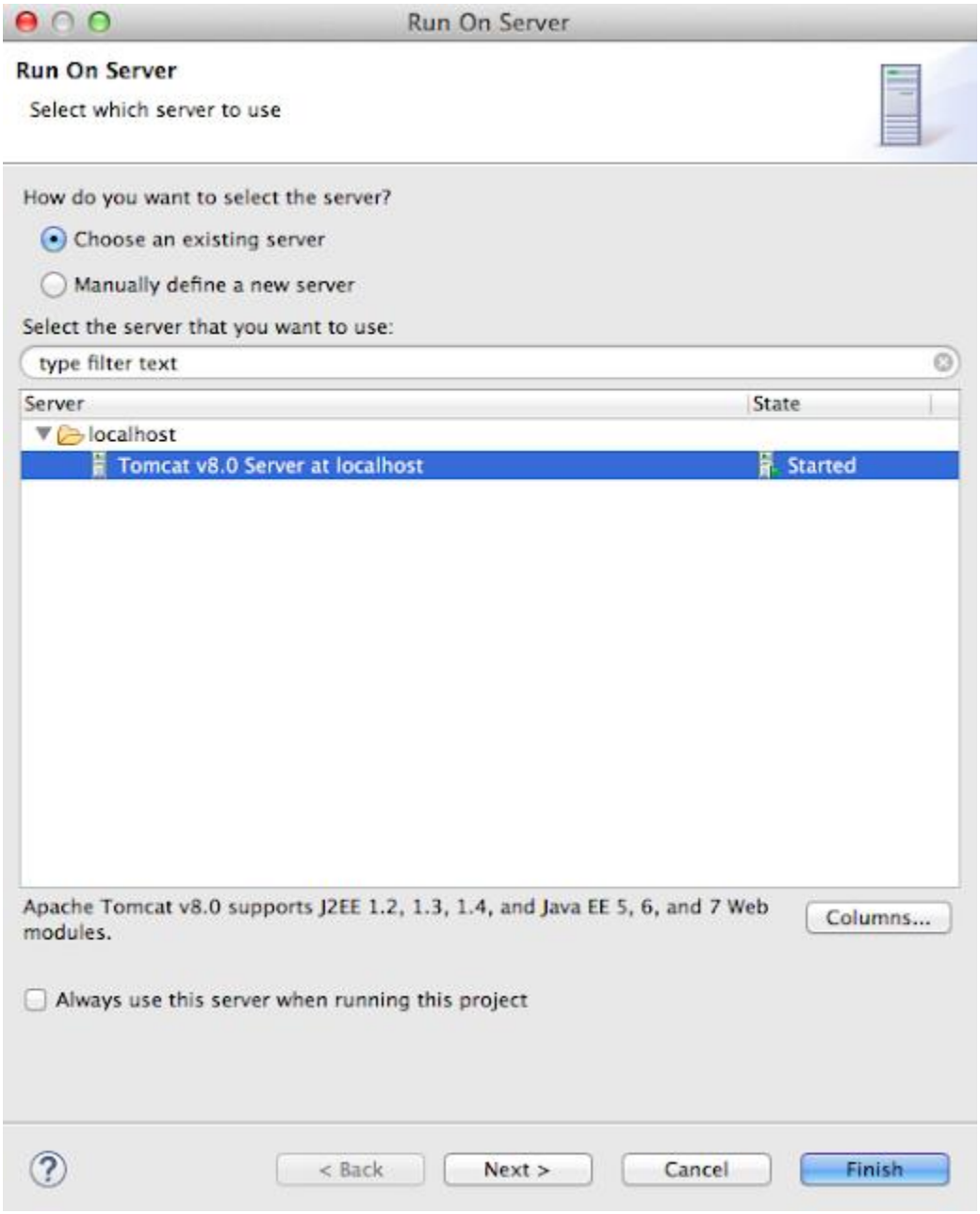


8) Provide goals as clean install (given below) and click on run



Run the application

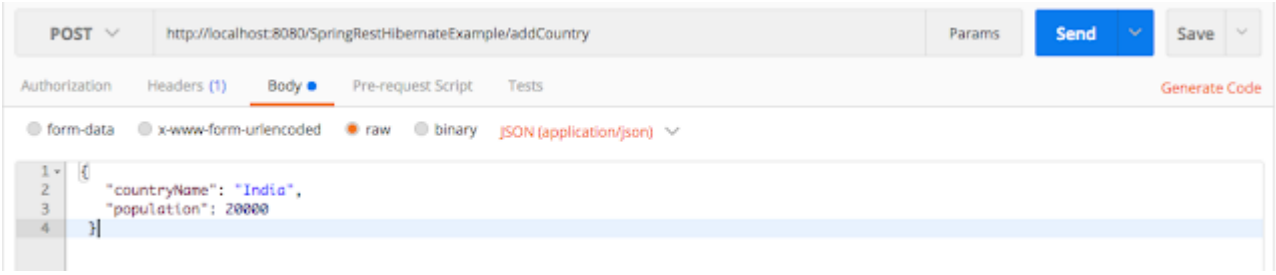
9) Right click on project -> run as -> run on server
Select apache tomcat and click on finish



10) We will test this application in postman , UI based client for testing restful web applications. It is chrome plugin. Launch postman.If you want java based client, then you can also use how to send get or post request in java.

Post method

12) Post method is used to create new resource. Here we are adding new Country India to country list, so you can see we have used new country json in post body.
URL: "http://localhost:8080/SpringRestHibernateExample/addCountry".



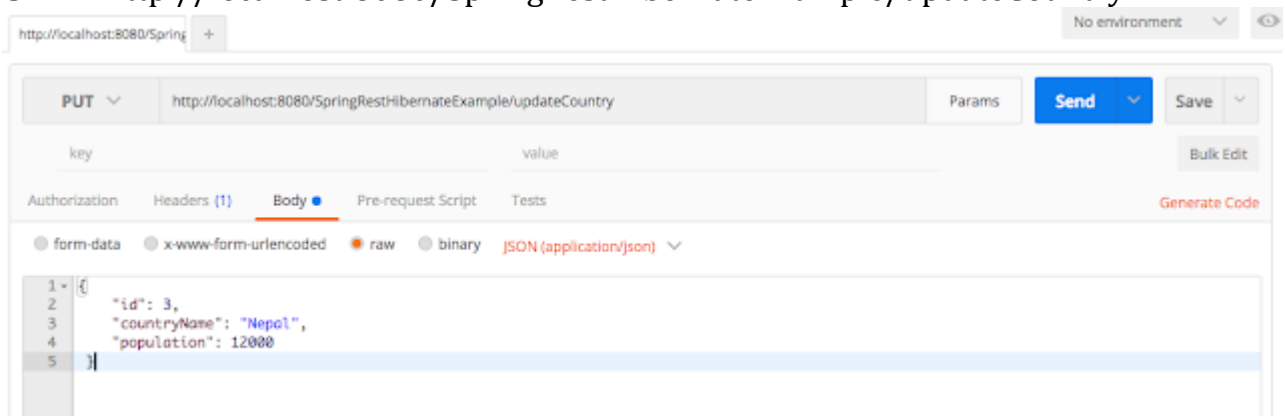
Lets see corresponding entry in Country table in database.

id	countryName	population
1	India	20000

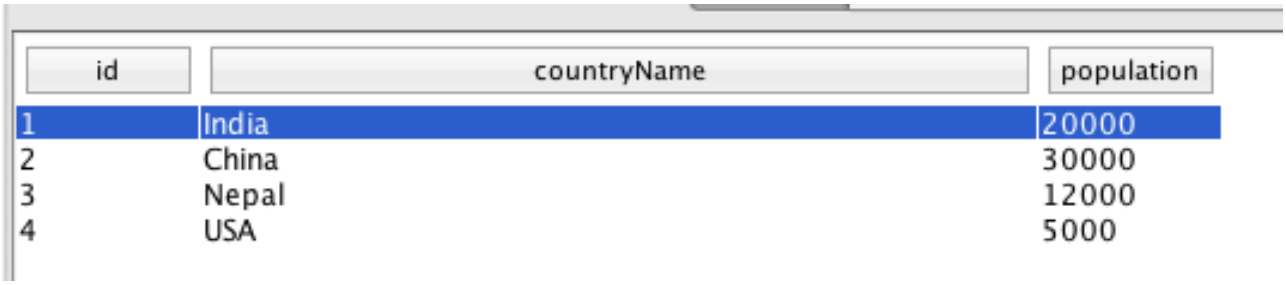
Lets create 3 more countries i.e. china, nepal and USA in similar way.

Put Method

13) Put method is used to update resource. Here will update population of nepal using put method. We will update country json in body of request.
URL : "http://localhost:8080/SpringRestHibernateExample/updateCountry"



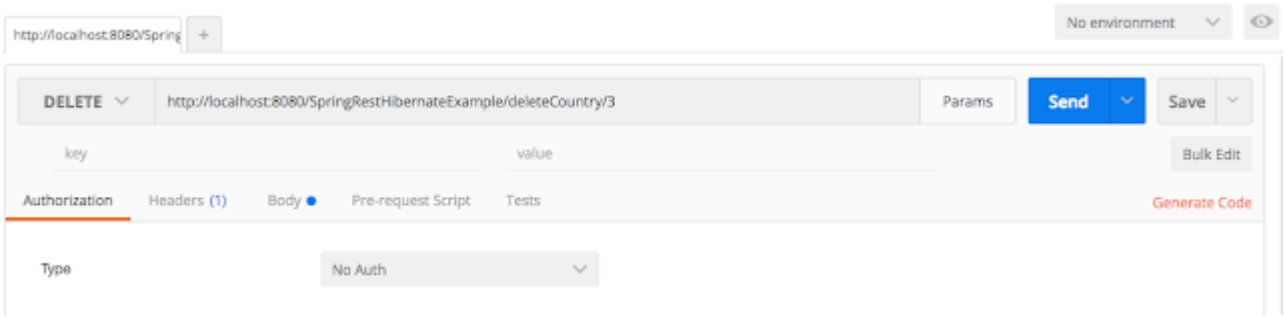
Lets check Nepal's population in database.



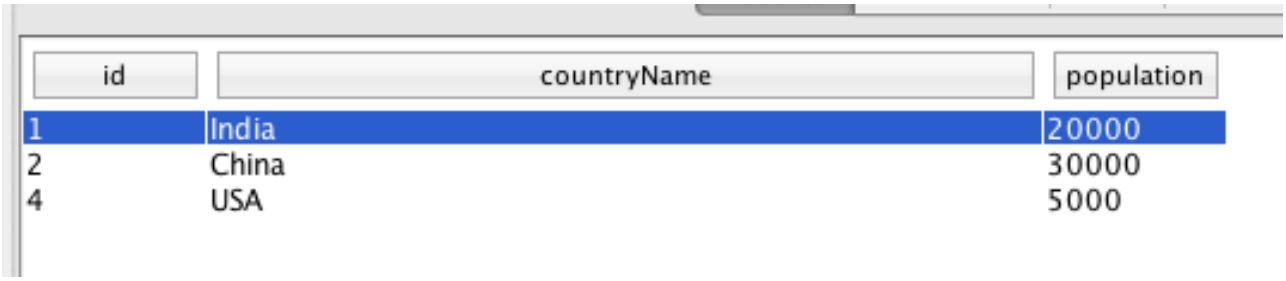
id	countryName	population
1	India	20000
2	China	30000
3	Nepal	12000
4	USA	5000

Delete method

14) Delete method is used to delete resource.We will pass id of country which needs to be deleted as PathParam. We are going delete id:3 i.e. Nepal to demonstrate delete method.
URL : "http://localhost:8080/SpringRestHibernateExample/deleteCountry/3"



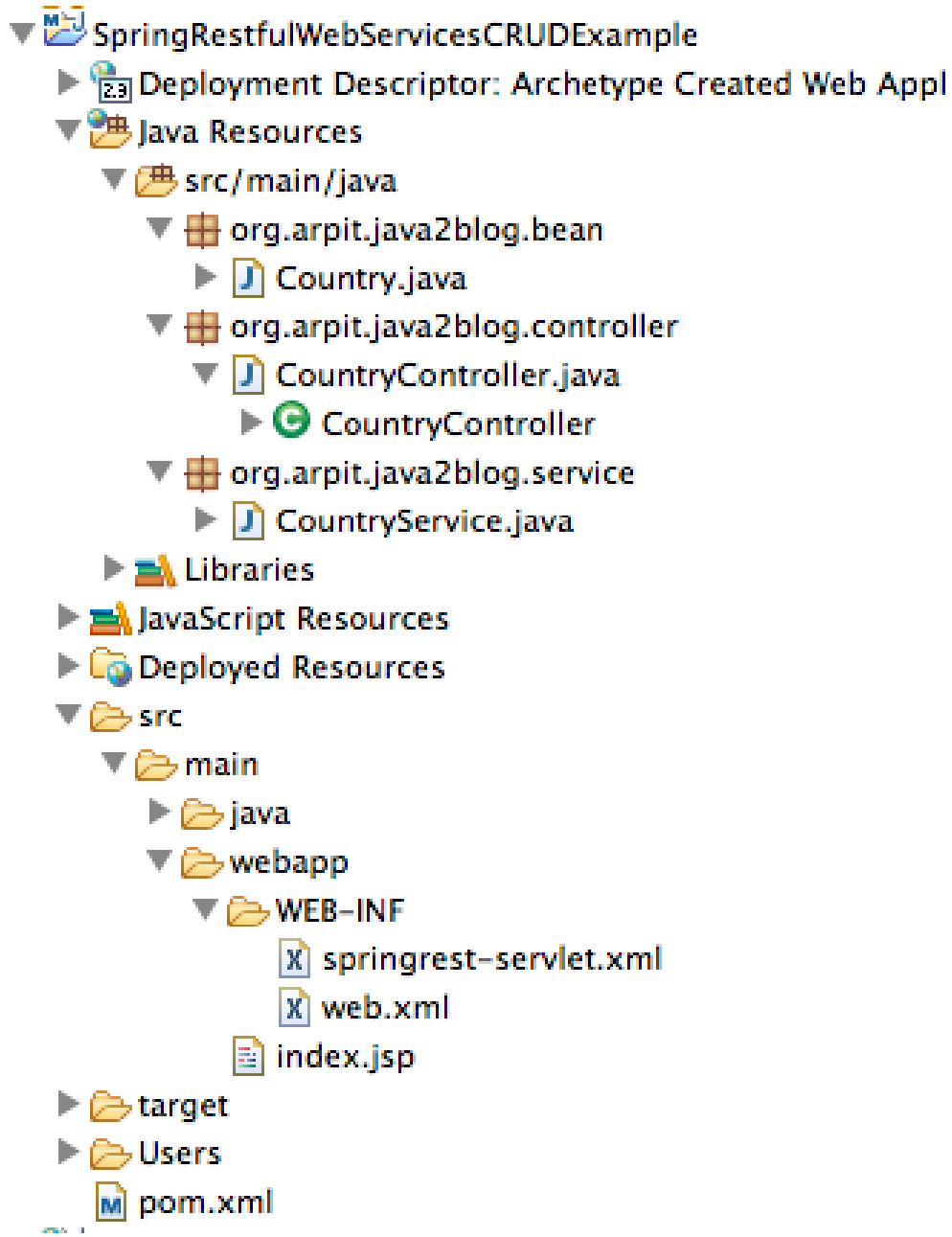
Lets check entry in database now.



id	countryName	population
1	India	20000
2	China	30000
4	USA	5000

As you can see, we have deleted **country with id 3 i.e. Nepal**

Project structure:

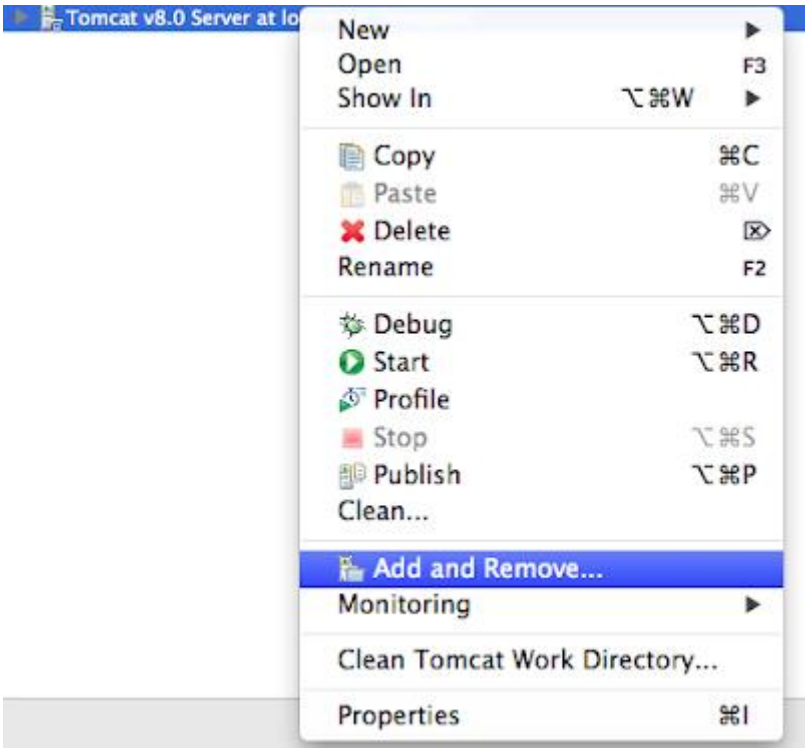


We are done with Spring Restful web services json CRUD example. If you are still facing any issue, please comment.

If you getting 404 error with above steps, you may need to follow below steps:

1) If you are getting this warning into your Tomcat startup console log, then it can cause the issue
WARNING: [SetPropertiesRule]{Server/Service/Engine/Host/Context} Setting property 'source' to 'org.eclipse.jst.j2ee.server: JAXRSJsonCRUDExample' did not find a matching property.

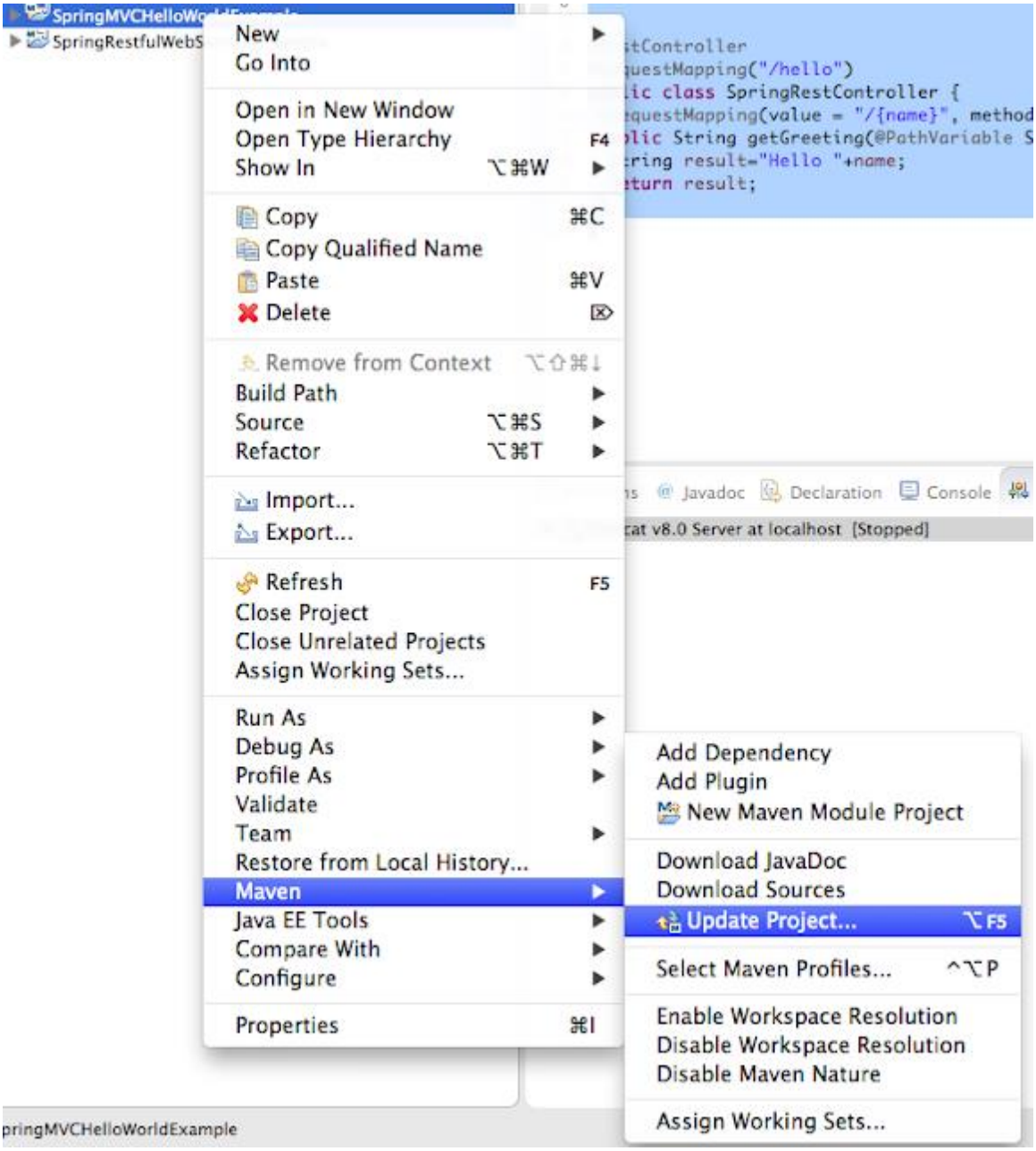
This particular warning basically means that the element in Tomcat's server.xml contains an unknown attribute source and that Tomcat doesn't know what to do with this attribute and therefore will ignore it.
To resolve this in eclipse,
Remove the project from the server from the Server View. Right click on server -> add and remove



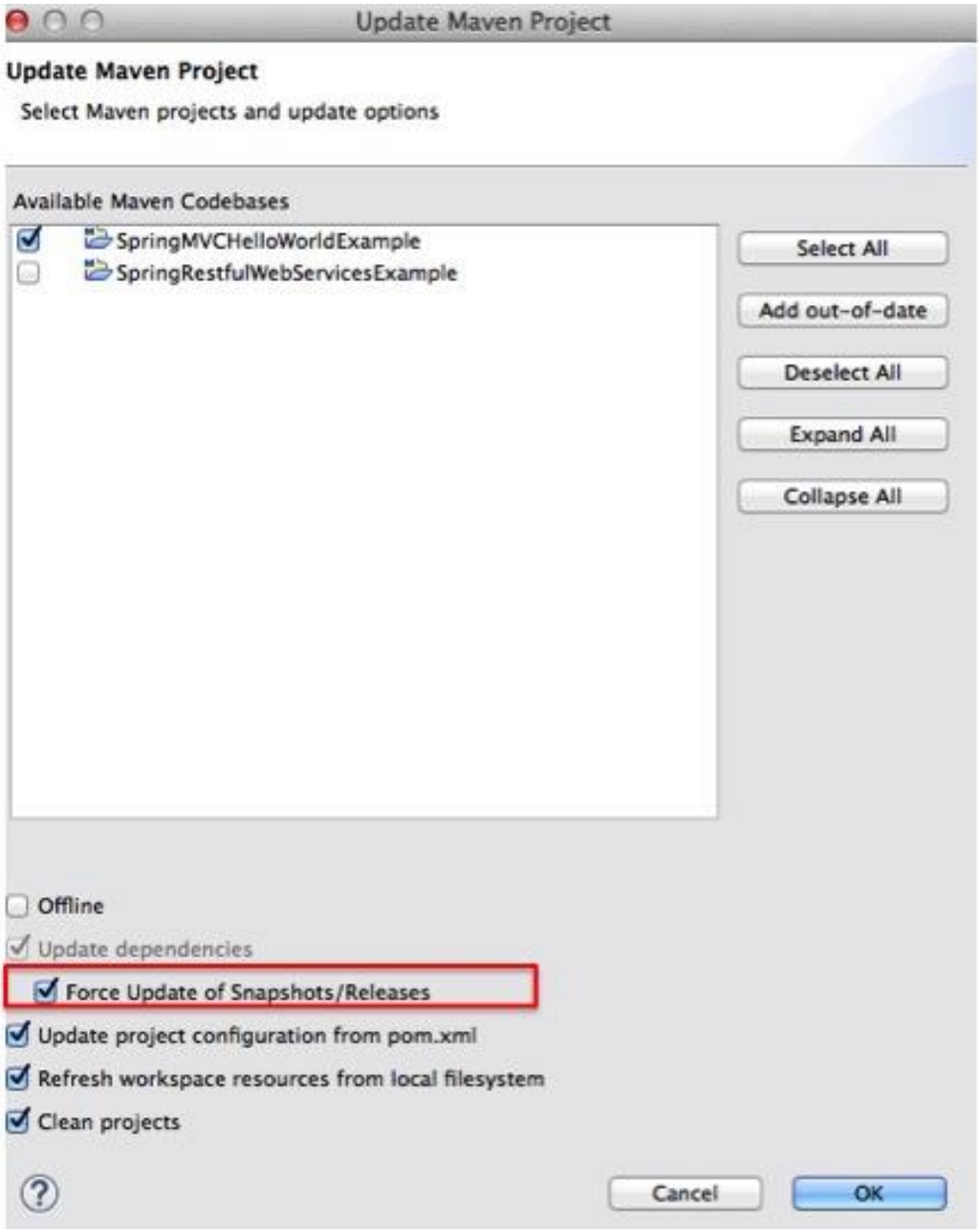
then remove project from server configuration.
Then run the project under the same server. Warning should be removed now
Or if warning still remains then

- Go to server view
- Double click on your tomcat server. It will open the server configuration.
- Under server options check 'Publish module contents to separate XML files' checkbox.
- Restart your server. This time your page will come without any issues.

2) Try to update Maven project.
Right click on project ->Maven-> update project


SpringMVCHelloWorldExample
SpringRestfulWebS
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Source ⌥⌘S
Refactor ⌥⌘T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Restore from Local History...
Maven
Java EE Tools
Compare With
Configure
Properties ⌘I
Add Dependency
Add Plugin
New Maven Module Project
Download JavaDoc
Download Sources
Update Project... ⌥F5
Select Maven Profiles... ^⌥P
Enable Workspace Resolution
Disable Workspace Resolution
Disable Maven Nature
Assign Working Sets...
Javadoc
Declaration
Console
at v8.0 Server at localhost [Stopped]
pringMVCHelloWorldExample

then



This should solve you issues.

AngularJS Restful web service example using $http::

In previous post, we have already seen Restful web services CRUD example. In this post, we will use AngularJS to call rest CRUD APIs. So we are going to create a view and then perform CRUD operations on the basis of button clicks. Source code:

Download

click to begin

20KB .zip

Here are steps to create a simple Restful web services(JAXWS) using jersey which will provide CRUD opertion.

1) Create a dynamic web project using maven in eclipse named "JAXRSJsonCRUDExample"

Maven dependencies

2) We need to add jersey jars utility in the classpath.

```
1
2  <dependency>
3    <groupId>com.sun.jersey</groupId>
4    <artifactId>jersey-servlet</artifactId>
5    <version>${jersey.version}</version>
6  </dependency>
7  <dependency>
8    <groupId>com.sun.jersey</groupId>
9    <artifactId>jersey-json</artifactId>
10   <version>${jersey.version}</version>
11 </dependency>
12
```

Jersey internally uses Jackson for Json Handling, so it will be used to marshal pojo objects to JSON.

Now create pom.xml as follows:

pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
3  nce"
4  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6  <groupId>com.arpit.java2blog</groupId>
7  <artifactId>JAXRSJsonExample</artifactId>
8  <packaging>war</packaging>
9  <version>0.0.1-SNAPSHOT</version>
10 <name>JAXRSJsonExample Maven Webapp</name>
11 <url>http://maven.apache.org</url>
```

```
12   <properties>
13       <jersey.version>1.18.3</jersey.version>
14    </properties>
15  <dependencies>
16   <dependency>
17    <groupId>junit</groupId>
18    <artifactId>junit</artifactId>
19    <version>3.8.1</version>
20    <scope>test</scope>
21   </dependency>
22
23   <dependency>
24    <groupId>com.sun.jersey</groupId>
25    <artifactId>jersey-servlet</artifactId>
26    <version>${jersey.version}</version>
27   </dependency>
28   <dependency>
29    <groupId>com.sun.jersey</groupId>
30    <artifactId>jersey-json</artifactId>
31    <version>${jersey.version}</version>
32   </dependency>
33
34   <dependency>
35    <groupId>commons-logging</groupId>
36    <artifactId>commons-logging</artifactId>
37    <version>1.2</version>
38   </dependency>
39  </dependencies>
40  <build>
41   <finalName>JAXRSJsonExample</finalName>
42   <plugins>
43    <plugin>
44     <groupId>org.apache.maven.plugins</groupId>
45     <artifactId>maven-compiler-plugin</artifactId>
46     <version>3.3</version>
```

```
47    <configuration>
48     <source>1.7</source>
49     <target>1.7</target>
50    </configuration>
51   </plugin>
52   <plugin>
53    <groupId>org.apache.maven.plugins</groupId>
54    <artifactId>maven-war-plugin</artifactId>
55    <configuration>
56     <failOnMissingWebXml>false</failOnMissingWebXml>
57    </configuration>
58   </plugin>
59  </plugins>
60 </build>
61
62 </project>
```

Application configuration:

3) create web.xml as below:

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns="http://java.sun.com/xml/ns/javaee"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd
6  "
7   version="3.0">
8   <display-name>Archetype Created Web Application</display-name>
9   <servlet>
10   <servlet-name>jersey-serlvet</servlet-name>
11   <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
12   <init-param>
13    <param-name>com.sun.jersey.config.property.packages</param-name>
14    <param-value>org.arpit.java2blog.controller</param-value>
15   </init-param>
16   <init-param>
```

```
17   <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
18   <param-value>true</param-value>
19  </init-param>
20
21  <load-on-startup>1</load-on-startup>
22 </servlet>
23 <servlet-mapping>
24  <servlet-name>jersey-serlvet</servlet-name>
25  <url-pattern>/rest/*</url-pattern>
26 </servlet-mapping>
27 </web-app>
```

Please change initParam "**com.sun.jersey.config.property.package**" property to provide correct controller package name if you are not using same package.

Create bean class

4) Create a bean name "Country.java" in org.arpit.java2blog.bean.

```
1
2  package org.arpit.java2blog.bean;
3
4  public class Country{
5
6          int id;
7          String countryName;
8          long population;
9
10         public Country() {
11                 super();
12         }
13         public Country(int i, String countryName,long population) {
14                 super();
15                 this.id = i;
16                 this.countryName = countryName;
17                 this.population=population;
18         }
```

```
19          public int getId() {
20                  return id;
21          }
22          public void setId(int id) {
23                  this.id = id;
24          }
25          public String getCountryName() {
26                  return countryName;
27          }
28          public void setCountryName(String countryName) {
29                  this.countryName = countryName;
30          }
31          public long getPopulation() {
32                  return population;
33          }
34          public void setPopulation(long population) {
35                  this.population = population;
36          }
37
38 }
39
```

Create Controller

5) Create a controller named “CountryController.java” in package **org.arpit.java2blog.controller**

```
 1
 2 package org.arpit.java2blog.controller;
 3
 4 import java.util.List;
 5
 6 import javax.ws.rs.DELETE;
 7 import javax.ws.rs.GET;
 8 import javax.ws.rs.POST;
 9 import javax.ws.rs.PUT;
10 import javax.ws.rs.Path;
11 import javax.ws.rs.PathParam;
```

```
12 import javax.ws.rs.Produces;
13 import javax.ws.rs.core.MediaType;
14
15 import org.arpit.java2blog.bean.Country;
16 import org.arpit.java2blog.service.CountryService;
17
18
19 @Path("/countries")
20 public class CountryController {
21
22          CountryService countryService=new CountryService();
23
24          @GET
25          @Produces(MediaType.APPLICATION_JSON)
26          public List getCountries()
27          {
28
29                  List listOfCountries=countryService.getAllCountries();
30                  return listOfCountries;
31          }
32
33          @GET
34          @Path("/{id}")
35          @Produces(MediaType.APPLICATION_JSON)
36          public Country getCountryById(@PathParam("id") int id)
37          {
38                  return countryService.getCountry(id);
39          }
40
41          @POST
42          @Produces(MediaType.APPLICATION_JSON)
43          public Country addCountry(Country country)
44          {
45                  return countryService.addCountry(country);
46          }
```

```
47
48          @PUT
49          @Produces(MediaType.APPLICATION_JSON)
50          public Country updateCountry(Country country)
51          {
52                  return countryService.updateCountry(country);
53
54          }
55
56          @DELETE
57          @Path("/{id}")
58          @Produces(MediaType.APPLICATION_JSON)
59          public void deleteCountry(@PathParam("id") int id)
60          {
61                  countryService.deleteCountry(id);
62
63          }
64
65 }
66
```

@Path(/your_path_at_class_level) : Sets the path to base URL + /your_path_at_class_level. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.
@Path(/your_path_at_method_level): Sets path to base URL + /your_path_at_class_level+ /your_path_at_method_level
@Produces(MediaType.APPLICATION_JSON[, more-types]): @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/json") is produced.

Create Service class

6) Create a class CountryService.java in package **org.arpit.java2blog.service**
It is just a helper class which should be replaced by database implementation. It is not very well written class, it is just used for demonstration.

```
1
2  package org.arpit.java2blog.service;
3
```

```
4  import java.util.ArrayList;
5  import java.util.HashMap;
6  import java.util.List;
7
8  import org.arpit.java2blog.bean.Country;
9  import org.arpit.java2blog.exception.CountryNotFoundException;
10
11 /*
12  * It is just a helper class which should be replaced by database implementation.
13  * It is not very well written class, it is just used for demonstration.
14  */
15 public class CountryService {
16
17          static HashMap<Integer,Country> countryIdMap=getCountryIdMap();
18
19
20          public CountryService() {
21                  super();
22
23                  if(countryIdMap==null)
24                  {
25                          countryIdMap=new HashMap<Integer,Country>();
26                          // Creating some object of countries while initializing
27                          Country indiaCountry=new Country(1, "India",10000);
28                          Country chinaCountry=new Country(4, "China",20000);
29                          Country nepalCountry=new Country(3, "Nepal",8000);
30                          Country bhutanCountry=new Country(2, "Bhutan",7000);
31
32
33                          countryIdMap.put(1,indiaCountry);
34                          countryIdMap.put(4,chinaCountry);
35                          countryIdMap.put(3,nepalCountry);
36                          countryIdMap.put(2,bhutanCountry);
37                  }
38          }
```

```
39
40          public List getAllCountries()
41          {
42                      List countries = new ArrayList(countryIdMap.values());
43                      return countries;
44          }
45
46          public Country getCountry(int id)
47          {
48                      Country country= countryIdMap.get(id);
49
50                      if(country == null)
51                      {
52                                  throw new CountryNotFoundException("Country with id "+id+" not found");
53                      }
54                      return country;
55          }
56          public Country addCountry(Country country)
57          {
58                      country.setId(getMaxId()+1);
59                      countryIdMap.put(country.getId(), country);
60                      return country;
61          }
62
63          public Country updateCountry(Country country)
64          {
65                      if(country.getId()<=0)
66                                  return null;
67                      countryIdMap.put(country.getId(), country);
68                      return country;
69
70          }
71          public void deleteCountry(int id)
72          {
73                      countryIdMap.remove(id);
```

```
74              }
75
76              public static HashMap<Integer, Country> getCountryIdMap() {
77                              return countryIdMap;
78              }
79
80              // Utility method to get max id
81              public static int getMaxId()
82              {   int max=0;
83              for (int id:countryIdMap.keySet()) {
84                              if(max<=id)
85                                              max=id;
86
87              }
88
89              return max;
90              }
91 }
92
```

AngularJS view

7) create **angularJSCrudExample.html** in WebContent folder with following content:

```
 1
 2  <html>
 3   <head>
 4
 5   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
 6
 7    <title>AngularJS $http Rest example</title>
 8  <script type="text/javascript">
 9          var app = angular.module("CountryManagement", []);
10
11          //Controller Part
12          app.controller("CountryController", function($scope, $http) {
13
```

```
14
15          $scope.countries = [];
16          $scope.countryForm = {
17            id : -1,
18            countryName : "",
19            population : ""
20          };
21
22          //Now load the data from server
23          _refreshCountryData();
24
25          //HTTP POST/PUT methods for add/edit country
26          // with the help of id, we are going to find out whether it is put or post operation
27
28          $scope.submitCountry = function() {
29
30            var method = "";
31            var url = "";
32            if ($scope.countryForm.id == -1) {
33              //Id is absent in form data, it is create new country operation
34              method = "POST";
35              url = 'rest/countries';
36            } else {
37              //Id is present in form data, it is edit country operation
38              method = "PUT";
39              url = 'rest/countries';
40            }
41
42            $http({
43              method : method,
44              url : url,
45              data : angular.toJson($scope.countryForm),
46              headers : {
47                'Content-Type' : 'application/json'
48              }
```

```
49              }).then( _success, _error );
50            };
51
52            //HTTP DELETE- delete country by Id
53            $scope.deleteCountry = function(country) {
54              $http({
55                method : 'DELETE',
56                url : 'rest/countries/' + country.id
57              }).then(_success, _error);
58            };
59
60          // In case of edit, populate form fields and assign form.id with country id
61            $scope.editCountry = function(country) {
62
63              $scope.countryForm.countryName = country.countryName;
64              $scope.countryForm.population = country.population;
65              $scope.countryForm.id = country.id;
66            };
67
68            /* Private Methods */
69            //HTTP GET- get all countries collection
70            function _refreshCountryData() {
71              $http({
72                method : 'GET',
73                url : 'http://localhost:8080/AngularjsJAXRSCRUDExample/rest/countries'
74              }).then(function successCallback(response) {
75                $scope.countries = response.data;
76              }, function errorCallback(response) {
77                console.log(response.statusText);
78              });
79            }
80
81            function _success(response) {
82              _refreshCountryData();
83              _clearFormData()
```

```
84              }
85
86              function _error(response) {
87                  console.log(response.statusText);
88              }
89
90              //Clear the form
91              function _clearFormData() {
92                  $scope.countryForm.id = -1;
93                  $scope.countryForm.countryName = "";
94                  $scope.countryForm.population = "";
95
96              };
97          });
98      </script>
99      <style>
100
101     .blue-button{
102     background: #25A6E1;
103     filter: progid: DXImageTransform.Microsoft.gradient( startColorstr='#25A6E1',endColorstr='#188BC0',Gradi
    entType=0);
104
105     padding:3px 5px;
106     color:#fff;
107     font-family:'Helvetica Neue',sans-serif;
108     font-size:12px;
109     border-radius:2px;
110     -moz-border-radius:2px;
111     -webkit-border-radius:4px;
112     border:1px solid #1A87B9
113         }
114
115      .red-button{
116     background: #CD5C5C;
117
118     padding:3px 5px;
        color:#fff;
```

```
font-family:'Helvetica Neue',sans-serif;
font-size:12px;
border-radius:2px;
-moz-border-radius:2px;
-webkit-border-radius:4px;
border:1px solid #CD5C5C
    }

  table {
    font-family: "Helvetica Neue", Helvetica, sans-serif;
    width: 50%;
  }

  caption {
    text-align: left;
    color: silver;
    font-weight: bold;
    text-transform: uppercase;
    padding: 5px;
  }

  th {
    background: SteelBlue;
    color: white;
  }


  tbody tr:nth-child(even) {
    background: WhiteSmoke;
  }

  tbody tr td:nth-child(2) {
    text-align:center;
  }

```

```
154     tbody tr td:nth-child(3),
155     tbody tr td:nth-child(4) {
156         text-align: center;
157         font-family: monospace;
158     }
159
160     tfoot {
161         background: SeaGreen;
162         color: white;
163         text-align: right;
164     }
165
166     tfoot tr th:last-child {
167         font-family: monospace;
168     }
169
170     td,th{
171          border: 1px solid gray;
172          width: 25%;
173          text-align: left;
174          padding: 5px 10px;
175      }
176
177
178
179      </style>
180  <head>
181  <body ng-app="CountryManagement" ng-controller="CountryController">
182       <h1>
183         AngularJS Restful web services example using $http
184       </h1>
185       <form ng-submit="submitCountry()">
186          <table>
187
188              <tr>
```

```
189             <th colspan="2">Add/Edit country</th>
190           </tr>
191           <tr>
192             <td>Country</td>
193             <td><input type="text" ng-model="countryForm.countryName" /></td>
194           </tr>
195           <tr>
196             <td>Population</td>
197             <td><input type="text" ng-model="countryForm.population"  /></td>
198           </tr>
199           <tr>
200             <td colspan="2"><input type="submit" value="Submit" class="blue-button" /></td>
201           </tr>
202         </table>
203       </form>
204       <table>
205         <tr>
206
207           <th>CountryName</th>
208           <th>Population</th>
209           <th>Operations</th>
210
211         </tr>
212
213         <tr ng-repeat="country in countries">
214
215   <td> {{ country.countryName }}</td>
216   <td >{{ country.population }}</td>
217
218             <td><a ng-click="editCountry(country)" class="blue-button">Edit</a> | <a ng-click="deleteCountry(co
      untry)" class="red-button">Delete</a></td>
219         </tr>
220
221       </table>
222
223
```

```
224
225
226
227   </body>
228 </html>
229
```

Explanation :

-
- We have injected $http as we have done in ajax example through controller constructor.

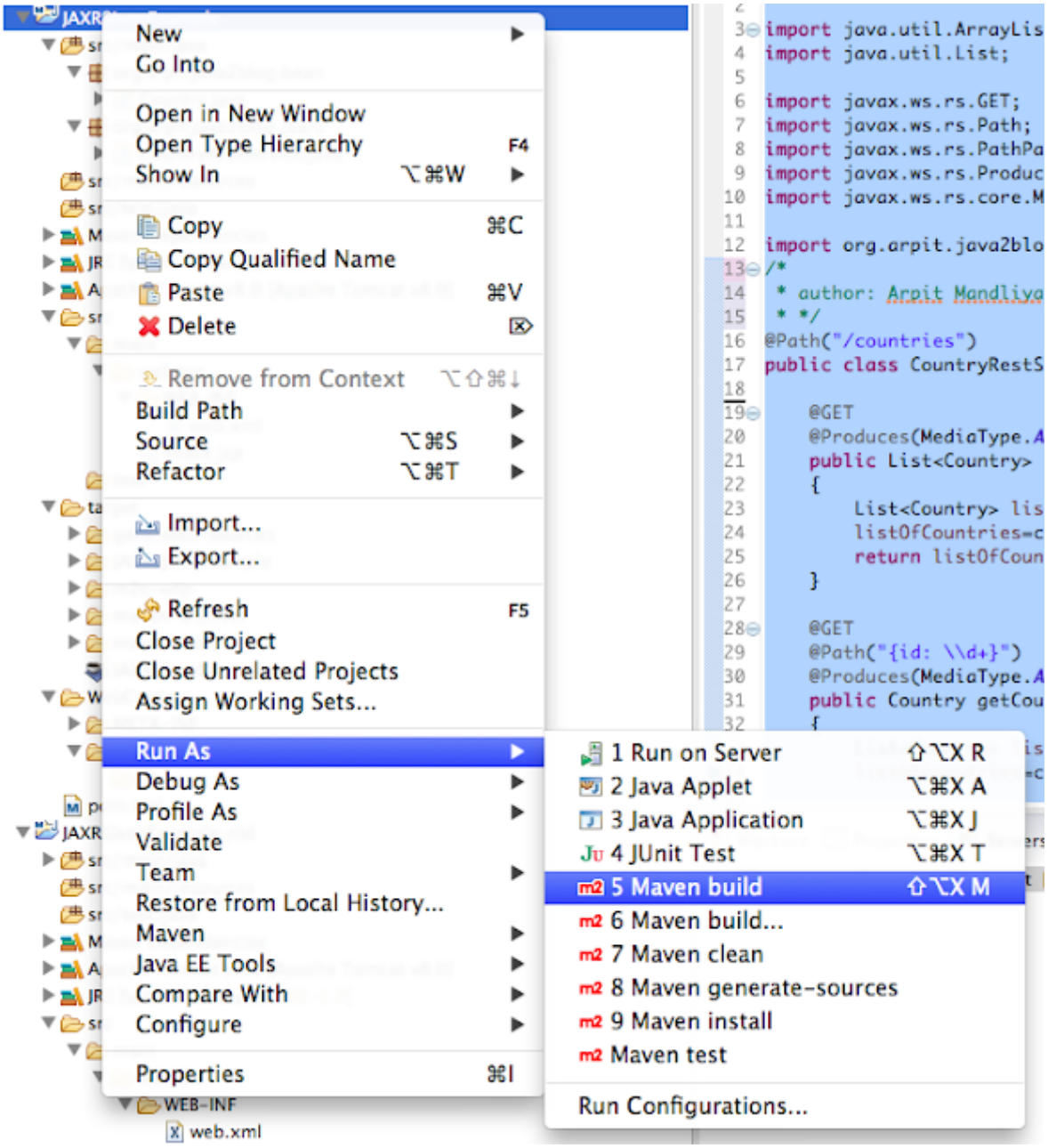
```
1
2 app.controller("CountryController", function($scope, $http) {
3
4
5           $scope.countries = [];
6 ...
7
```

-
- We have defined various methods depending on operations such as editCountry, deleteCountry, submitCountry
- When you click on submit button on form, it actually calls POST or PUT depending on operation. If you click on edit and submit data then it will be put operation as it will be update on existing resource. If you directly submit data, then it will be POST operation to create new resource,
- Every time you submit data, it calls refereshCountryData() to refresh country table below.
- When you call $http, you need to pass method type and URL, it will call it according, You can either put absolute URL or relative URL with respect to context root of web application.

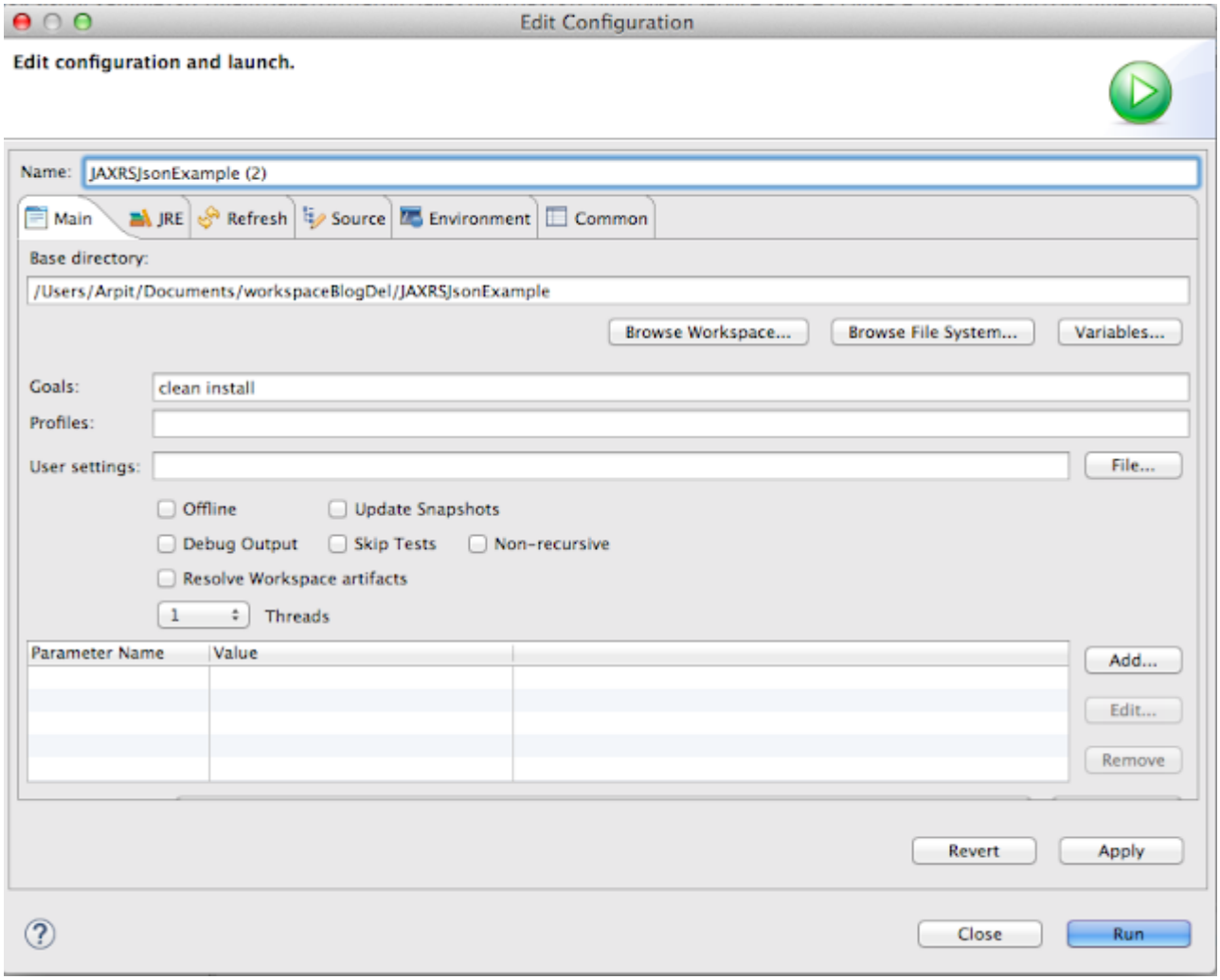
```
1
2   //HTTP GET- get all countries collection
3             function _refreshCountryData() {
4                 $http({
5                   method : 'GET',
6                   url : 'http://localhost:8080/JAXRSJsonCRUDExample/rest/countries'
7                 }).then(function successCallback(response) {
```

```
8                $scope.countries = response.data;
9             }, function errorCallback(response) {
10               console.log(response.statusText);
11            });
12        }
13
```

8) It 's time to do maven build.
Right click on project -> Run as -> Maven build



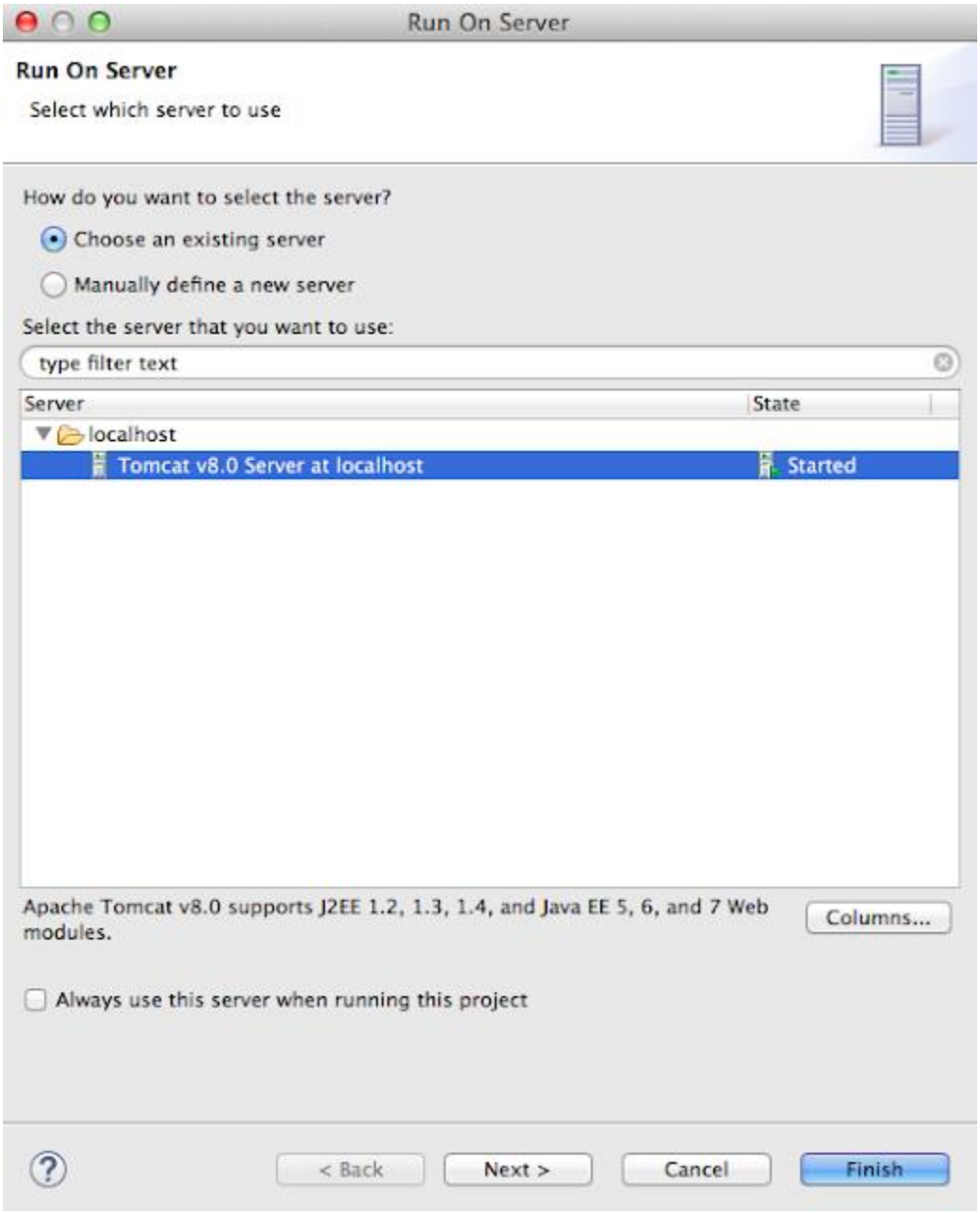
9) Provide goals as clean install (given below) and click on run



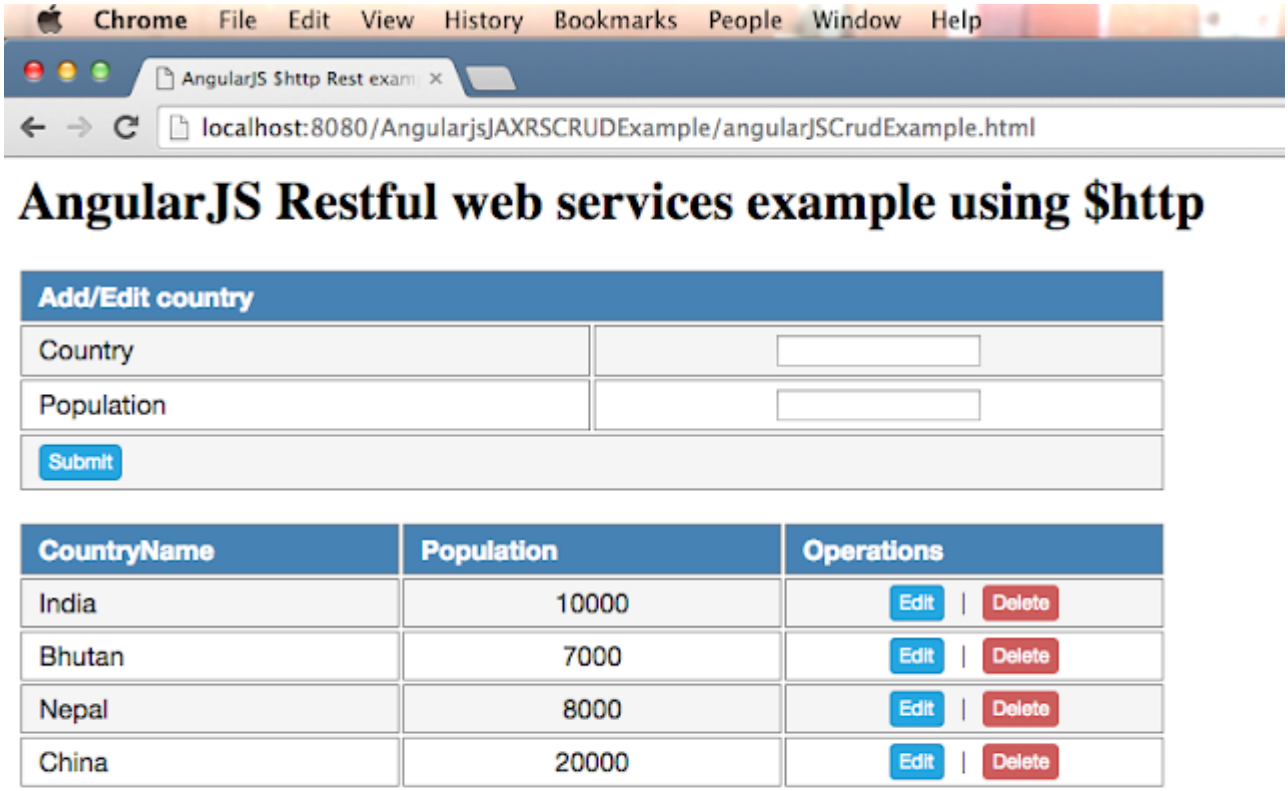
Run the application

10) Right click on **angularJSCrudExample.html** -> run as -> run on server

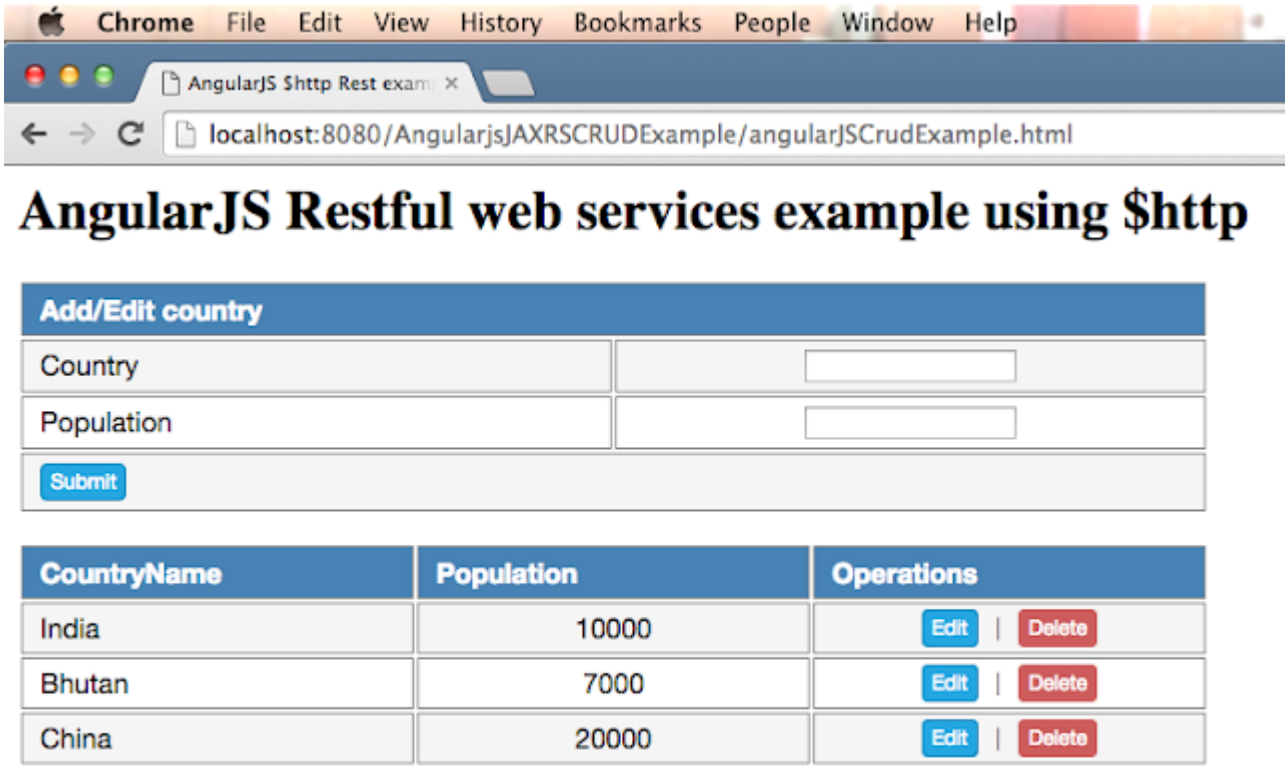
Select apache tomcat and click on finish



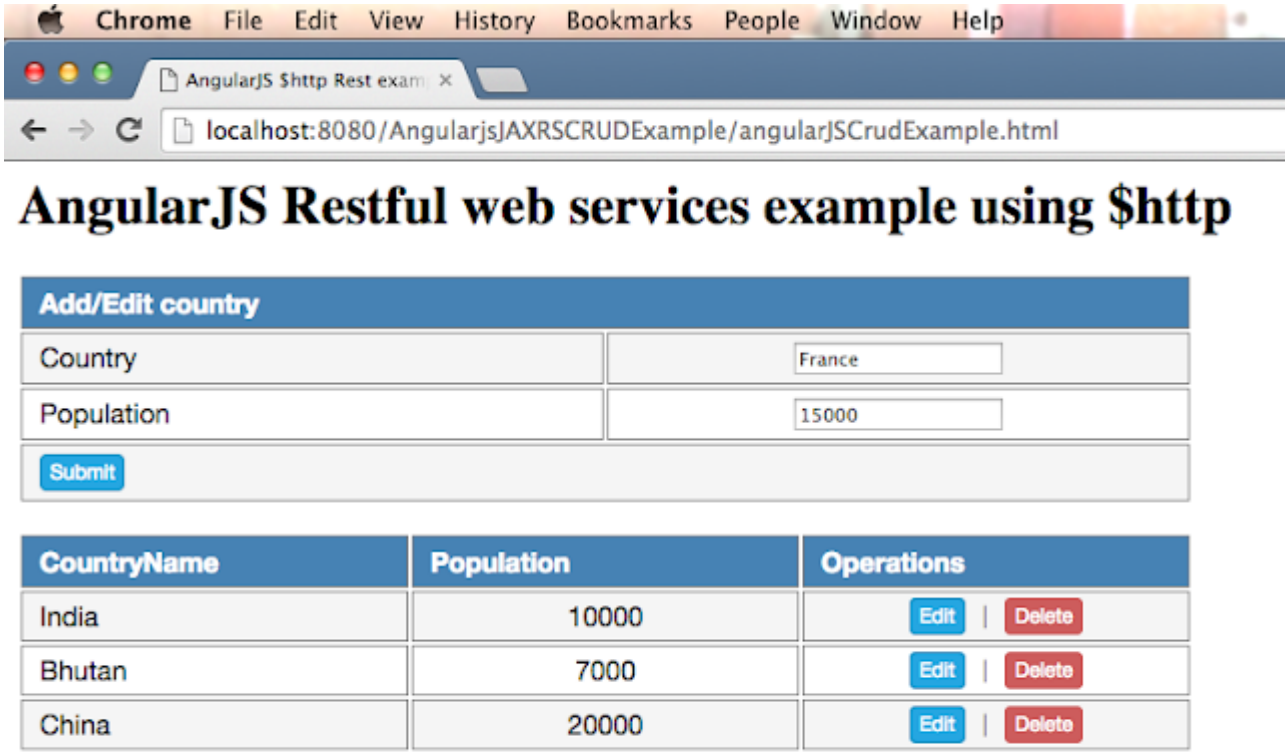
10) You should be able to see below page URL :
"http://localhost:8080/AngularjsJAXRSCRUDExample/angularJSCrudExample.html"



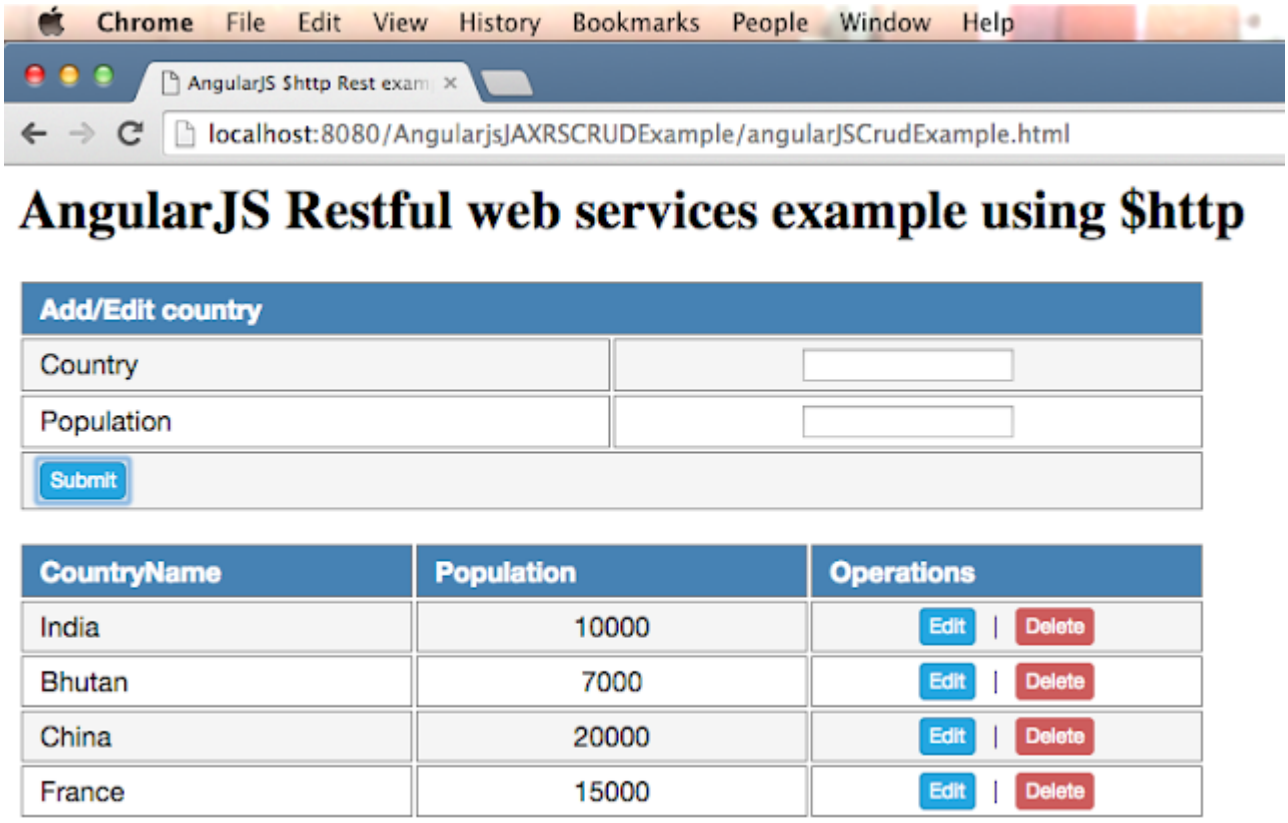
Lets click on delete button corresponding to Nepal and you will see below screen:



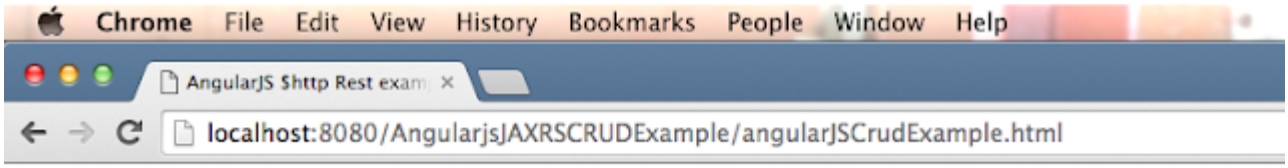
Lets add new country France with population 15000



Click on submit and you will see below screen.



Now click on edit button corresponding to India and change population from 10000 to 100000.

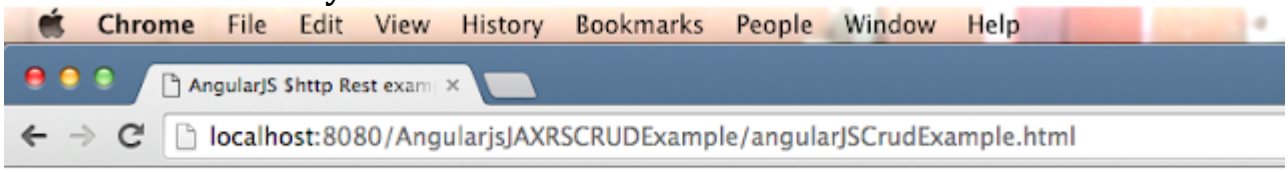


AngularJS Restful web services example using $http

Add/Edit country	
Country	India
Population	100000
Submit	

CountryName	Population	Operations
India	10000	Edit | Delete
Bhutan	7000	Edit | Delete
China	20000	Edit | Delete
France	15000	Edit | Delete

Click on submit and you will see below screen:



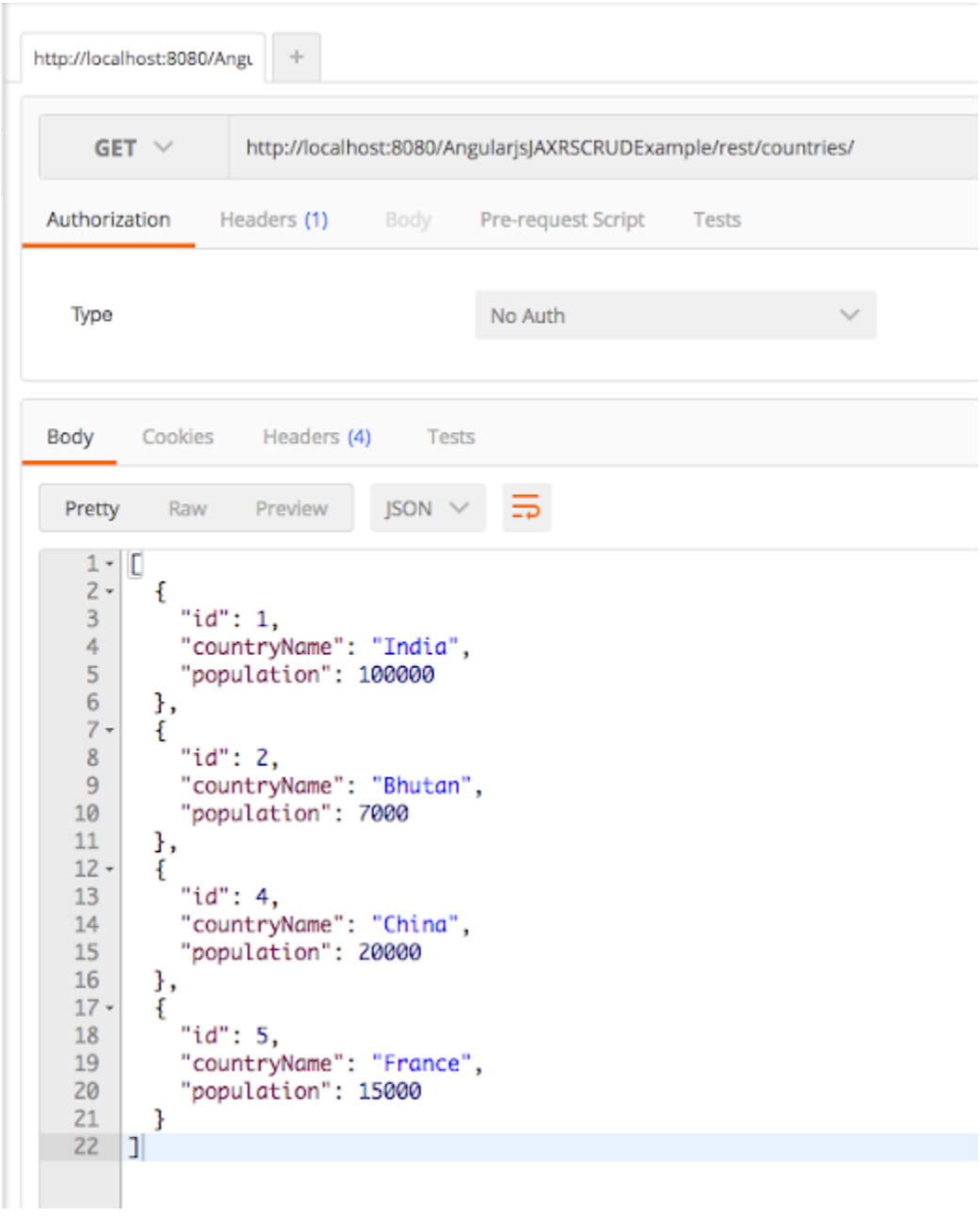
AngularJS Restful web services example using $http

Add/Edit country	
Country	
Population	
Submit	

CountryName	Population	Operations
India	100000	Edit | Delete
Bhutan	7000	Edit | Delete
China	20000	Edit | Delete
France	15000	Edit | Delete

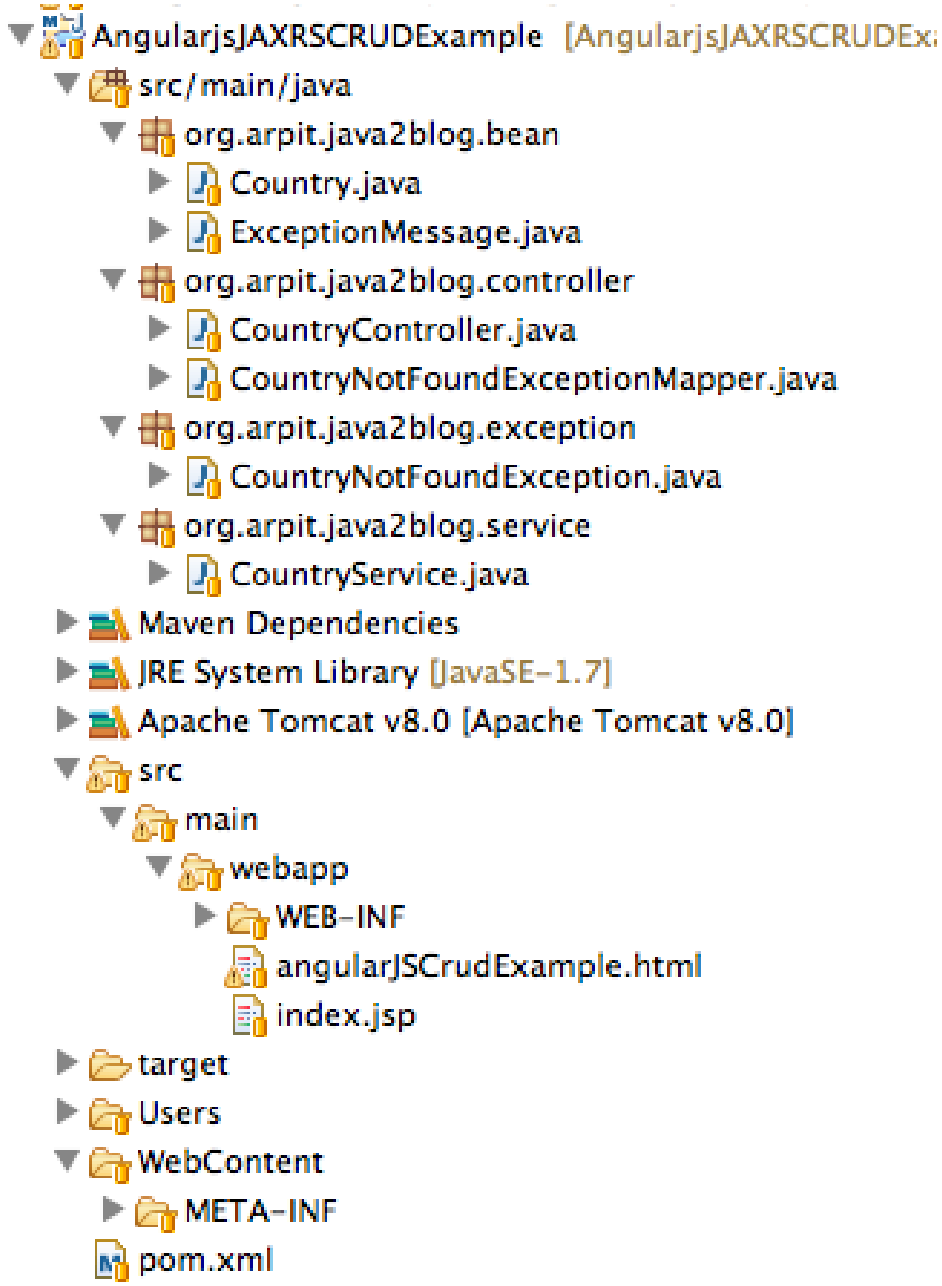
Lets check Get method for Rest API

11) Test your get method REST service **URL :**"http://localhost:8080/AngularjsJAXRSCRUDExample/rest/countries/". You will get following output:



As you can see , all changes have been reflected in above get call.

Project structure:



We are done with Restful web services json CRUD example using jersey.

Spring MVC Hibernate MySQL CRUD example::

In this post, we are going to see integration of Spring MVC, hibernate and mysql CRUD example.
We have already seen integration of Spring Rest with hibernate in previous tutorial.

Here are steps to create a project with Spring MVC , hibernate and mySQL crud example.

Github Source code:

Download

1) Create a dynamic web project using maven in eclipse named "SpringMVCHibernateCRUDExample"

Maven dependencies

2) We are using Spring 4 and Hibernate 4 for this application. It should work with hibernate 3 also but we need to do small changes in hibernate configuration.

pom.xml

```
1
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
3  tance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.x
5  sd">
6          <modelVersion>4.0.0</modelVersion>
7          <groupId>com.arpit.java2blog</groupId>
8          <artifactId>SpringMVCHibernateCRUDExample</artifactId>
9          <packaging>war</packaging>
10         <version>0.0.1-SNAPSHOT</version>
11         <name>SpringMVCHibernateCRUDExample Maven Webapp</name>
12         <url>http://maven.apache.org</url>
13         <dependencies>
14                 <dependency>
15                         <groupId>junit</groupId>
16                         <artifactId>junit</artifactId>
17                         <version>3.8.1</version>
18                         <scope>test</scope>
19                 </dependency>
20                 <dependency>
21                         <groupId>javax.servlet</groupId>
22                         <artifactId>javax.servlet-api</artifactId>
23                         <version>3.1.0</version>
24                 </dependency>
25
26                 <dependency>
27                         <groupId>org.springframework</groupId>
28                         <artifactId>spring-core</artifactId>
29                         <version>${spring.version}</version>
30                 </dependency>
31                 <dependency>
32                         <groupId>org.springframework</groupId>
33                         <artifactId>spring-webmvc</artifactId>
34                         <version>${spring.version}</version>
35                 </dependency>
36                 <dependency>
37                         <groupId>com.fasterxml.jackson.core</groupId>
38                         <artifactId>jackson-databind</artifactId>
39                         <version>2.4.1</version>
40                 </dependency>
41                 <!-- Hibernate -->
42                 <dependency>
43                         <groupId>org.hibernate</groupId>
44                         <artifactId>hibernate-core</artifactId>
45                         <version>${hibernate.version}</version>
```

```
46                  </dependency>
47                  <dependency>
48                          <groupId>org.hibernate</groupId>
49                          <artifactId>hibernate-entitymanager</artifactId>
50                          <version>${hibernate.version}</version>
51                  </dependency>
52
53                  <!-- Apache Commons DBCP -->
54                  <dependency>
55                          <groupId>commons-dbcp</groupId>
56                          <artifactId>commons-dbcp</artifactId>
57                          <version>1.4</version>
58                  </dependency>
59                  <!-- Spring ORM -->
60                  <dependency>
61                          <groupId>org.springframework</groupId>
62                          <artifactId>spring-orm</artifactId>
63                          <version>${spring.version}</version>
64                  </dependency>
65
66                  <!-- AspectJ -->
67                  <dependency>
68                          <groupId>org.aspectj</groupId>
69                          <artifactId>aspectjrt</artifactId>
70                          <version>${org.aspectj-version}</version>
71                  </dependency>
72                  <dependency>
73                          <groupId>mysql</groupId>
74                          <artifactId>mysql-connector-java</artifactId>
75                          <version>5.1.6</version>
76                  </dependency>
77                  <!-- https://mvnrepository.com/artifact/jstl/jstl -->
78                  <dependency>
79                          <groupId>jstl</groupId>
80                          <artifactId>jstl</artifactId>
81                          <version>1.2</version>
82                  </dependency>
83          </dependencies>
84          <build>
85                  <finalName>SpringMVCHibernateCRUDExample</finalName>
86
87                  <plugins>
88                          <plugin>
89                                  <groupId>org.apache.maven.plugins</groupId>
90                                  <artifactId>maven-compiler-plugin</artifactId>
91                                  <version>3.1</version>
92                                  <configuration>
93                                          <source>${jdk.version}</source>
94                                          <target>${jdk.version}</target>
95                                  </configuration>
96                          </plugin>
97                          <plugin>
98                                  <groupId>org.apache.maven.plugins</groupId>
99                                  <artifactId>maven-war-plugin</artifactId>
100                                 <configuration>
101                                         <failOnMissingWebXml>false</failOnMissingWebXml>
102                                 </configuration>
103                         </plugin>
104                 </plugins>
105
106         </build>
107         <properties>
108                 <spring.version>4.2.1.RELEASE</spring.version>
109                 <security.version>4.0.3.RELEASE</security.version>
110                 <jdk.version>1.7</jdk.version>
```

```
111                     <hibernate.version>4.3.5.Final</hibernate.version>
112                     <org.aspectj-version>1.7.4</org.aspectj-version>
113             </properties>
    </project>
```

Spring application configuration:

3) Change web.xml as below:

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4             xmlns="http://java.sun.com/xml/ns/javaee"
5             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-ap
6  p_3_0.xsd"
7             version="3.0">
8             <display-name>Archetype Created Web Application</display-name>
9             <context-param>
10                    <param-name>contextConfigLocation</param-name>
11                    <param-value>/WEB-INF/applicationcontext.xml</param-value>
12            </context-param>
13            <listener>
14    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
15 </listener>
16            <servlet>
17  <servlet-name>spring</servlet-name>
18  <servlet-class>
19   org.springframework.web.servlet.DispatcherServlet
20  </servlet-class>
21  <load-on-startup>1</load-on-startup>
22 </servlet>
23
24 <servlet-mapping>
25  <servlet-name>spring</servlet-name>
26  <url-pattern>/</url-pattern>
27 </servlet-mapping>
28
29 </web-app>
```

4) create a xml file named spring-servlet.xml in /WEB-INF/ folder. **Please change context:component-scan if you want to use different package for spring to search for controller.Please refer to [spring mvc hello world example] for more understanding.**

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <beans:beans xmlns="http://www.springframework.org/schema/mvc"
4             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.springframe
5  work.org/schema/beans"
6             xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springfra
7  mework.org/schema/tx"
8             xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.o
9  rg/schema/mvc/spring-mvc.xsd
10                    http://www.springframework.org/schema/beans http://www.springframework.org/schema
11 /beans/spring-beans.xsd
12                    http://www.springframework.org/schema/context http://www.springframework.org/sche
13 ma/context/spring-context.xsd
14                    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/
15 spring-tx-4.0.xsd">
16
17            <annotation-driven />
18
19            <resources mapping="/resources/**" location="/resources/" />
20
```

```
21          <beans:bean
22                  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
23                  <beans:property name="prefix" value="/WEB-INF/views/" />
24                  <beans:property name="suffix" value=".jsp" />
25          </beans:bean>
26
27          <beans:bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
28                  destroy-method="close">
29                  <beans:property name="driverClassName" value="com.mysql.jdbc.Driver" />
30                  <beans:property name="url"
31                          value="jdbc:mysql://localhost:3306/CountryData" />
32                  <beans:property name="username" value="root" />
33                  <beans:property name="password" value="" />
34          </beans:bean>
35
36          <!-- Hibernate 4 SessionFactory Bean definition -->
37          <beans:bean id="hibernate4AnnotatedSessionFactory"
38                  class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
39                  <beans:property name="dataSource" ref="dataSource" />
40                  <beans:property name="annotatedClasses">
41                          <beans:list>
42                                  <beans:value>org.arpit.java2blog.model.Country</beans:value>
43                          </beans:list>
44                  </beans:property>
45                  <beans:property name="hibernateProperties">
46                          <beans:props>
47                                  <beans:prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect
48                                  </beans:prop>
49                                  <beans:prop key="hibernate.show_sql">true</beans:prop>
50                          </beans:props>
51                  </beans:property>
52          </beans:bean>
53
54          <context:component-scan base-package="org.arpit.java2blog" />
55
56          <tx:annotation-driven transaction-manager="transactionManager" />
57
58          <beans:bean id="transactionManager"
                    class="org.springframework.orm.hibernate4.HibernateTransactionManager">
                    <beans:property name="sessionFactory"
                            ref="hibernate4AnnotatedSessionFactory" />
            </beans:bean>
    </beans:beans>
```

In Spring-servlet.xml, we have done hibernate configuration. **dataSource** bean is used to specify java data source. We need to provide driver, URL , Username and Password. **transactionManager** bean is used to configure hibernate transaction manager. **hibernate4AnnotatedSessionFactory** bean is used to configure FactoryBean that creates a Hibernate SessionFactory. This is the common way to set up a shared Hibernate SessionFactory in a Spring application context, so you can use this SessionFactory to inject in Hibernate data access objects. Create a applicationcontext.xml in WEB-INF folder, this file is used for bean configuration as we are using spring-servlet.xml for bean configuration , we will keep this file empty.

```
1
2 <!--?xml version="1.0" encoding="UTF-8"?-->
3
```

Create bean class

4) Create a bean name “Country.java” in org.arpit.java2blog.bean.

```
 1
 2  package org.arpit.java2blog.model;
 3
 4  import javax.persistence.Column;
 5  import javax.persistence.Entity;
 6  import javax.persistence.GeneratedValue;
 7  import javax.persistence.GenerationType;
 8  import javax.persistence.Id;
 9  import javax.persistence.Table;
10
11  /*
12   * This is our model class and it corresponds to Country table in database
13   */
14  @Entity
15  @Table(name="COUNTRY")
16  public class Country{
17
18          @Id
19          @Column(name="id")
20          @GeneratedValue(strategy=GenerationType.IDENTITY)
21          int id;
22
23          @Column(name="countryName")
24          String countryName;
25
26          @Column(name="population")
27          long population;
28
29          public Country() {
30                  super();
31          }
32          public Country(int i, String countryName,long population) {
33                  super();
34                  this.id = i;
35                  this.countryName = countryName;
36                  this.population=population;
37          }
38          public int getId() {
39                  return id;
40          }
41          public void setId(int id) {
42                  this.id = id;
43          }
44          public String getCountryName() {
45                  return countryName;
46          }
47          public void setCountryName(String countryName) {
48                  this.countryName = countryName;
49          }
50          public long getPopulation() {
51                  return population;
52          }
53          public void setPopulation(long population) {
54                  this.population = population;
55          }
56
57  }
58
```

@Entity is used for making a persistent pojo class.For this java class,you will have corresponding table in database. @Column is used to map annotated

attribute to corresponding column in table. So Create Country table in mysql database with following code:

```
1
2 CREATE TABLE COUNTRY
3 (
4   id int PRIMARY KEY NOT NULL AUTO_INCREMENT,
5   countryName varchar(100) NOT NULL,
6   population int NOT NULL
7 )
8 ;
9
```

Create Controller

5) Create a controller named “CountryController.java” in package **org.arpit.java2blog.controller**

```
1
2  package org.arpit.java2blog.controller;
3
4
5  import java.util.List;
6
7  import org.arpit.java2blog.model.Country;
8  import org.arpit.java2blog.service.CountryService;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.ui.Model;
12 import org.springframework.web.bind.annotation.ModelAttribute;
13 import org.springframework.web.bind.annotation.PathVariable;
14 import org.springframework.web.bind.annotation.RequestBody;
15 import org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.bind.annotation.RequestMethod;
17
18 @Controller
19 public class CountryController {
20
21          @Autowired
22          CountryService countryService;
23
24          @RequestMapping(value = "/getAllCountries", method = RequestMethod.GET, headers = "Accept=appli
25 cation/json")
26          public String getCountries(Model model) {
27
28                  List listOfCountries = countryService.getAllCountries();
29                  model.addAttribute("country", new Country());
30                  model.addAttribute("listOfCountries", listOfCountries);
31                  return "countryDetails";
32          }
33
34          @RequestMapping(value = "/getCountry/{id}", method = RequestMethod.GET, headers = "Accept=appli
35 cation/json")
36          public Country getCountryById(@PathVariable int id) {
37                  return countryService.getCountry(id);
38          }
39
40          @RequestMapping(value = "/addCountry", method = RequestMethod.POST, headers = "Accept=applicat
41 ion/json")
42          public String addCountry(@ModelAttribute("country") Country country) {
43                  if(country.getId()==0)
44                  {
45                          countryService.addCountry(country);
46                  }
47                  else
```

```
48                      {
49                              countryService.updateCountry(country);
50                      }
51
52                      return "redirect:/getAllCountries";
53              }
54
55              @RequestMapping(value = "/updateCountry/{id}", method = RequestMethod.GET, headers = "Accept=a
56  pplication/json")
57              public String updateCountry(@PathVariable("id") int id,Model model) {
58                      model.addAttribute("country", this.countryService.getCountry(id));
59                      model.addAttribute("listOfCountries", this.countryService.getAllCountries());
60                      return "countryDetails";
61              }
62
63              @RequestMapping(value = "/deleteCountry/{id}", method = RequestMethod.GET, headers = "Accept=a
64  pplication/json")
65              public String deleteCountry(@PathVariable("id") int id) {
66                      countryService.deleteCountry(id);
                        return "redirect:/getAllCountries";

                }
    }
```

Create DAO class

Create a class **CountryDAO.java** in package **org.arpit.java2blog.dao.** This class will execute hibernate statements while interacting with database.

```
 1
 2  package org.arpit.java2blog.dao;
 3
 4  import java.util.List;
 5
 6  import org.arpit.java2blog.model.Country;
 7  import org.hibernate.Session;
 8  import org.hibernate.SessionFactory;
 9  import org.springframework.beans.factory.annotation.Autowired;
10  import org.springframework.stereotype.Repository;
11
12  @Repository
13  public class CountryDAO {
14
15              @Autowired
16              private SessionFactory sessionFactory;
17
18              public void setSessionFactory(SessionFactory sf) {
19                      this.sessionFactory = sf;
20              }
21
22              public List getAllCountries() {
23                      Session session = this.sessionFactory.getCurrentSession();
24                      List countryList = session.createQuery("from Country").list();
25                      return countryList;
26              }
27
28              public Country getCountry(int id) {
29                      Session session = this.sessionFactory.getCurrentSession();
30                      Country country = (Country) session.load(Country.class, new Integer(id));
31                      return country;
32              }
33
34              public Country addCountry(Country country) {
35                      Session session = this.sessionFactory.getCurrentSession();
36                      session.persist(country);
```

```
37                      return country;
38              }
39
40              public void updateCountry(Country country) {
41                      Session session = this.sessionFactory.getCurrentSession();
42                      session.update(country);
43              }
44
45              public void deleteCountry(int id) {
46                      Session session = this.sessionFactory.getCurrentSession();
47                      Country p = (Country) session.load(Country.class, new Integer(id));
48                      if (null != p) {
49                              session.delete(p);
50                      }
51              }
52 }
53
```

@Repository is specialised component annotation which is used to create bean at DAO layer. We have use Autowired annotation to inject hibernate SessionFactory into CountryDAO class. We have already configured hibernate SessionFactory object in Spring-Servlet.xml file.

Create Service class

6) Create a class CountryService.java in package **org.arpit.java2blog.service**

It is service level class. It will call DAO layer class.

```
1
2  package org.arpit.java2blog.service;
3
4  import java.util.List;
5  import org.arpit.java2blog.dao.CountryDAO;
6  import org.arpit.java2blog.model.Country;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.stereotype.Service;
9  import org.springframework.transaction.annotation.Transactional;
10
11 @Service("countryService")
12 public class CountryService {
13
14              @Autowired
15              CountryDAO countryDao;
16
17              @Transactional
18              public List getAllCountries() {
19                      return countryDao.getAllCountries();
20              }
21
22              @Transactional
23              public Country getCountry(int id) {
24                      return countryDao.getCountry(id);
25              }
26
27              @Transactional
28              public void addCountry(Country country) {
29                      countryDao.addCountry(country);
30              }
31
32              @Transactional
33              public void updateCountry(Country country) {
34                      countryDao.updateCountry(country);
35
36              }
```

```
37
38              @Transactional
39              public void deleteCountry(int id) {
40                              countryDao.deleteCountry(id);
41              }
42 }
43
```

@Service is specialised component annotation which is used to create bean at Service layer.

Create view

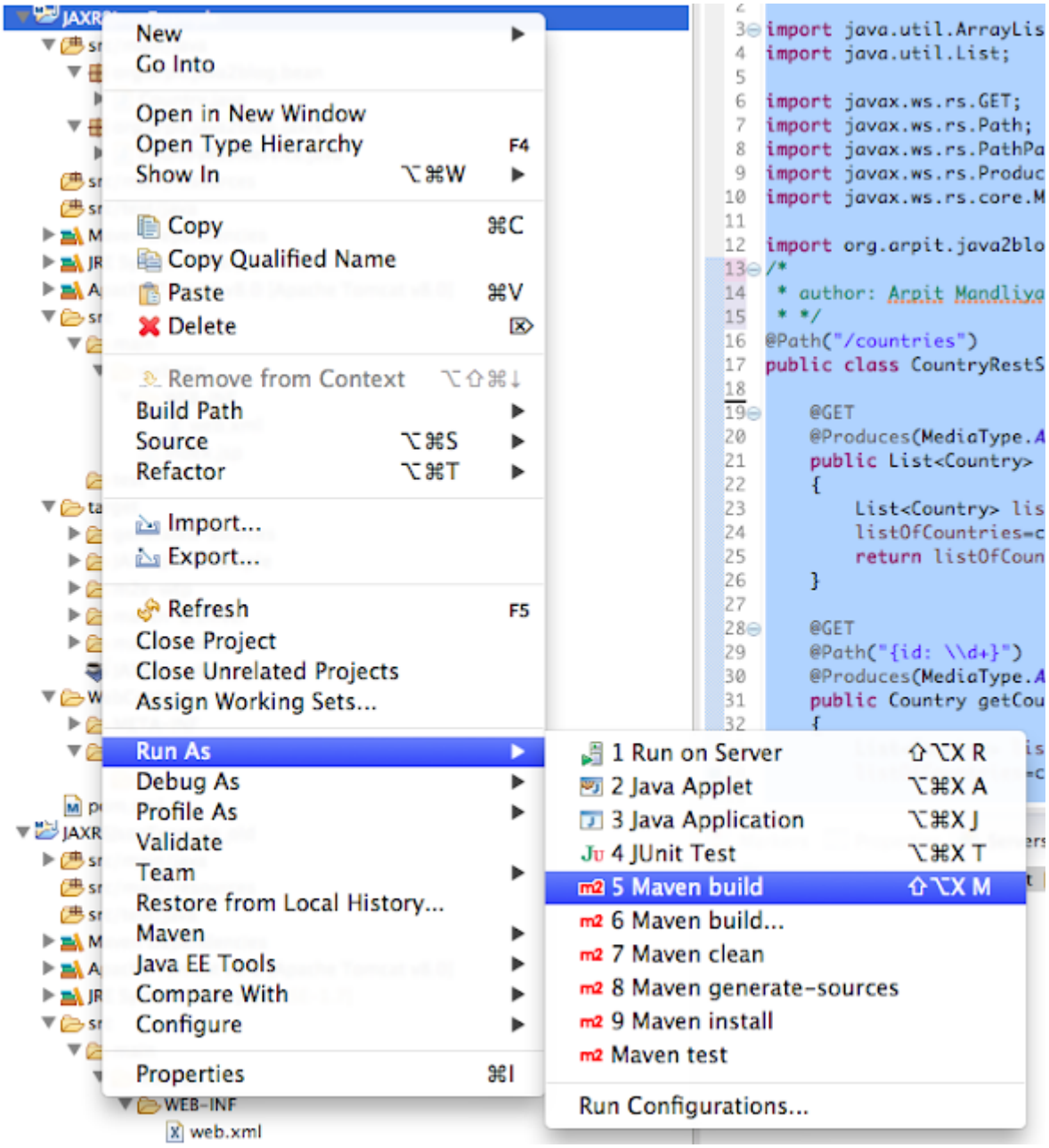
Create view called **countryDetails.jsp in WEB-INF/view/ folder.**

```
 1
 2  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
 3  <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
 4  <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
 5  <html>
 6  <head>
 7  <style>
 8  .blue-button{
 9              background: #25A6E1;
10              filter: progid: DXImageTransform.Microsoft.gradient( startColorstr='#25A6E1',endColorstr='#188BC0',
11 GradientType=0);
12              padding:3px 5px;
13              color:#fff;
14              font-family:'Helvetica Neue',sans-serif;
15              font-size:12px;
16              border-radius:2px;
17              -moz-border-radius:2px;
18              -webkit-border-radius:4px;
19              border:1px solid #1A87B9
20 }
21 table {
22   font-family: "Helvetica Neue", Helvetica, sans-serif;
23    width: 50%;
24 }
25 th {
26   background: SteelBlue;
27   color: white;
28 }
29  td,th{
30              border: 1px solid gray;
31              width: 25%;
32              text-align: left;
33              padding: 5px 10px;
34          }
35 </style>
36 </head>
37 <body>
38 <form:form method="post" modelAttribute="country" action="/SpringMVCHibernateCRUDExample/addCountr
39 y">
40 <table>
41                          <tr>
42                                  <th colspan="2">Add Country</th>
43                          </tr>
44                          <tr>
45              <form:hidden path="id" />
46         <td><form:label path="countryName">Country Name:</form:label></td>
47         <td><form:input path="countryName" size="30" maxlength="30"></form:input></td>
48        </tr>
49                          <tr>
50                                  <td><form:label path="population">Population:</form:label></td>
51         <td><form:input path="population" size="30" maxlength="30"></form:input></td>
```

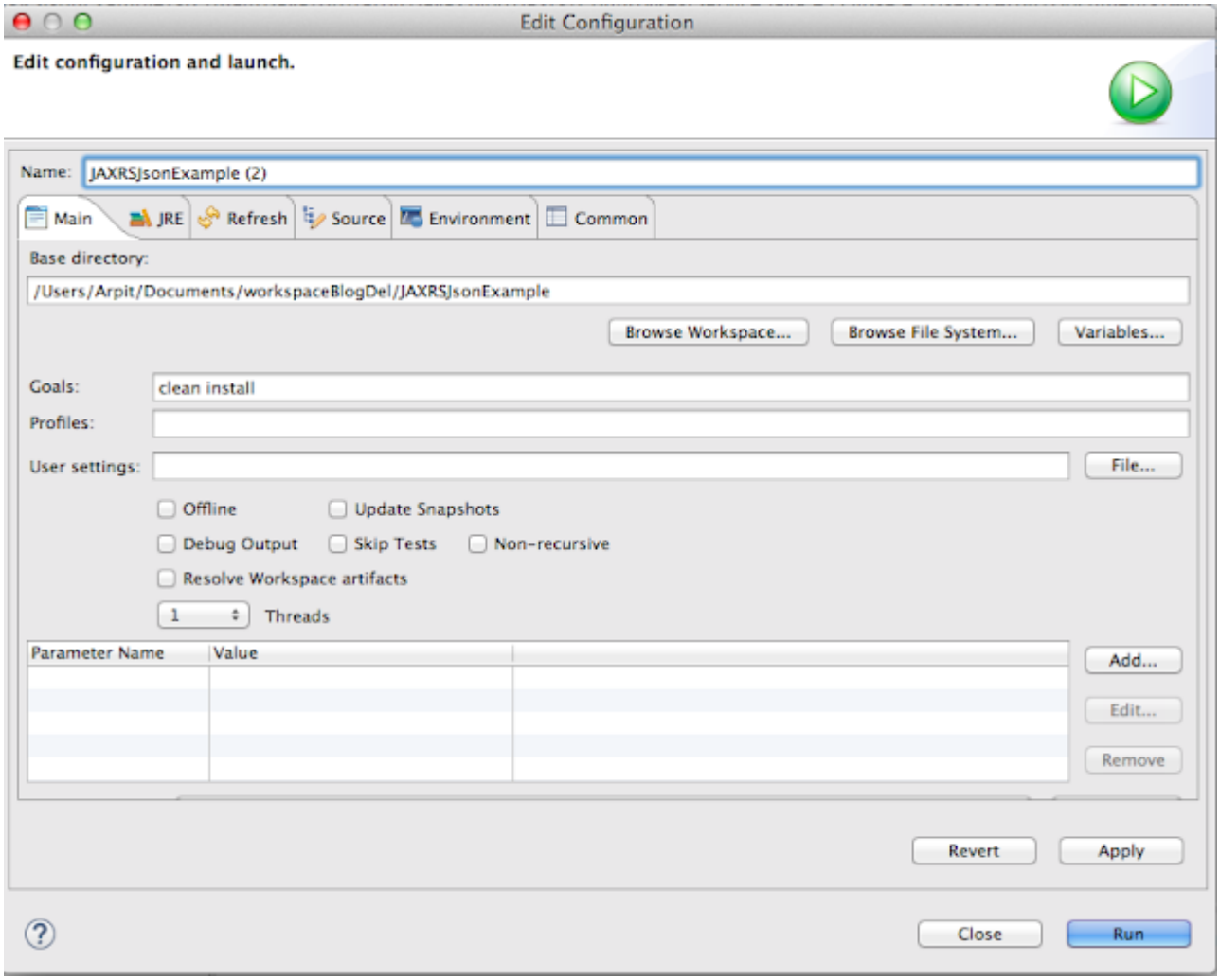
```
52                      </tr>
53                      <tr>
54                              <td colspan="2"><input type="submit"
55                                      class="blue-button" /></td>
56                      </tr>
57              </table>
58 </form:form>
59 </br>
60 <h3>Country List</h3>
61 <c:if test="${!empty listOfCountries}">
62              <table class="tg">
63              <tr>
64                      <th width="80">Id</th>
65                      <th width="120">Country Name</th>
66                      <th width="120">Population</th>
67                      <th width="60">Edit</th>
68                      <th width="60">Delete</th>
69              </tr>
70              <c:forEach items="${listOfCountries}" var="country">
71                      <tr>
72                              <td>${country.id}</td>
73                              <td>${country.countryName}</td>
74                              <td>${country.population}</td>
75                              <td><a href="<c:url value='/updateCountry/${country.id}' />" >Edit</a></td>
76                              <td><a href="<c:url value='/deleteCountry/${country.id}' />" >Delete</a></td>
77                      </tr>
78              </c:forEach>
79              </table>
80 </c:if>
81 </body>
   </html>
```

8) It ‘s time to do maven build.

Right click on project -> Run as -> Maven build

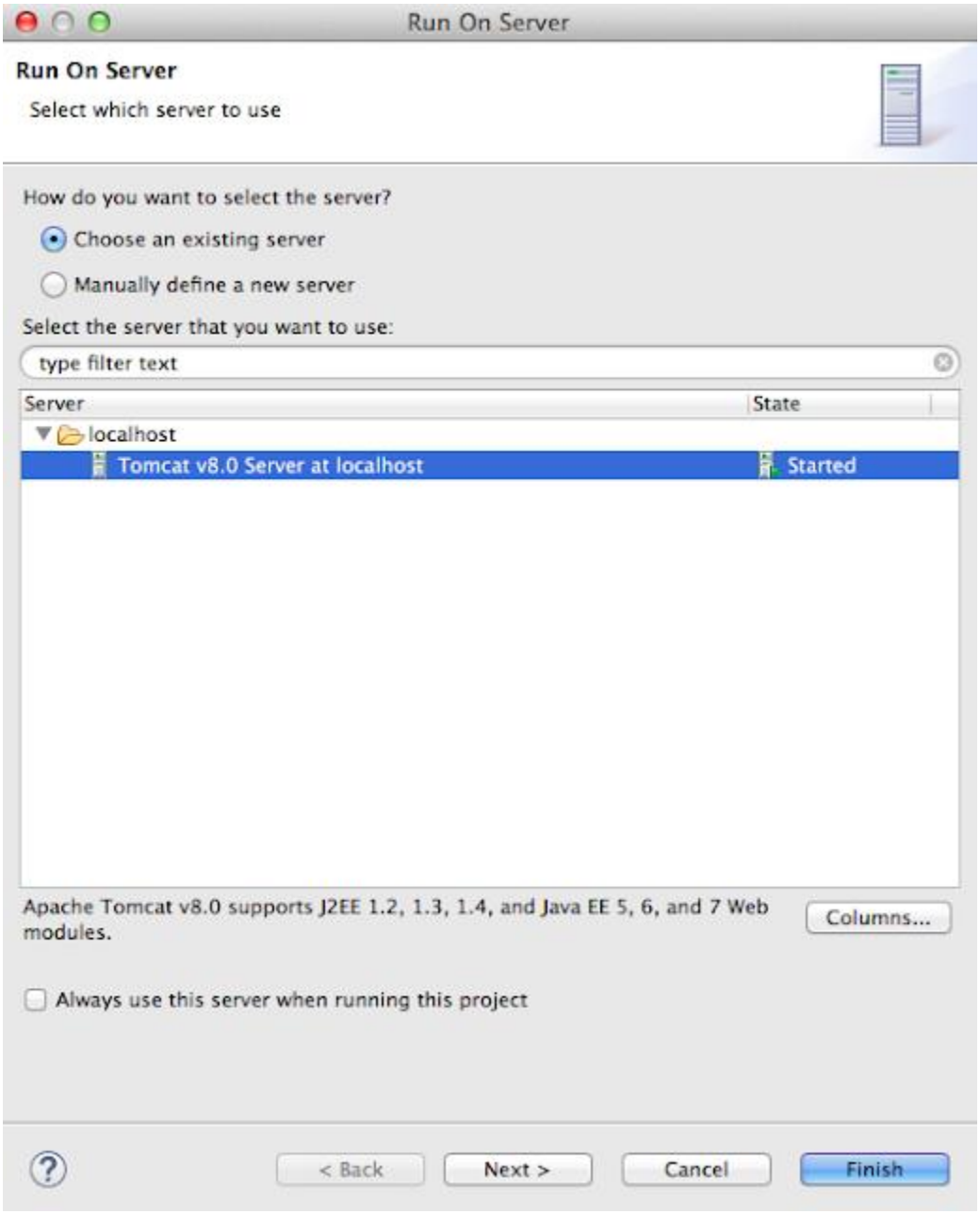


9) Provide goals as clean install (given below) and click on run



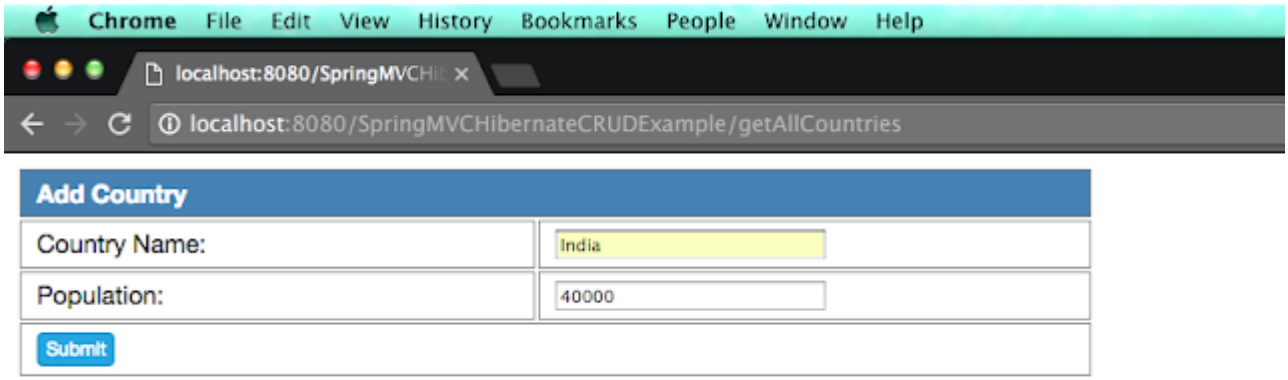
Run the application

10) Right click on project -> run as -> run on server
Select apache tomcat and click on finish



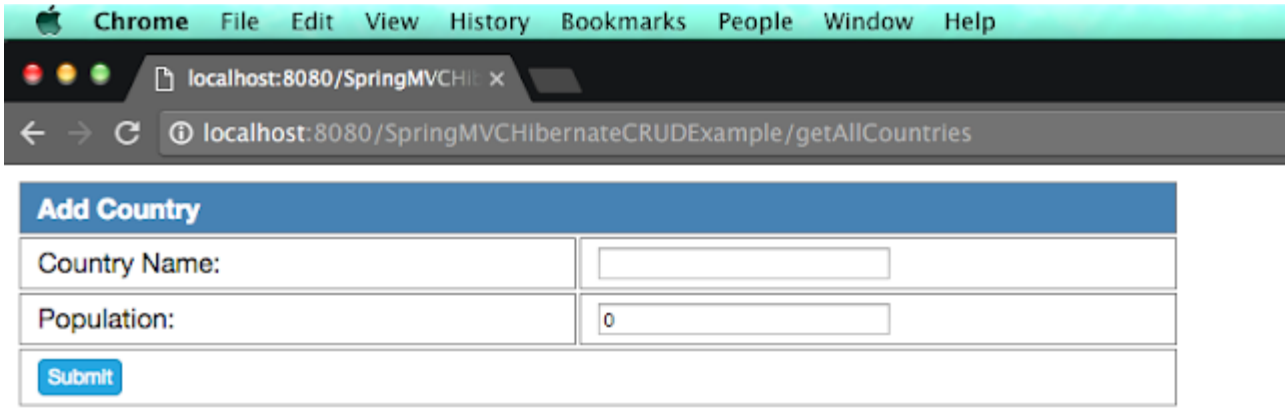
11) Now lets hit below URL to getAllCountries. **http://localhost:8080/SpringMVCHibernateCRUDExample/getAllCountries**

You will get below screen:



Country List

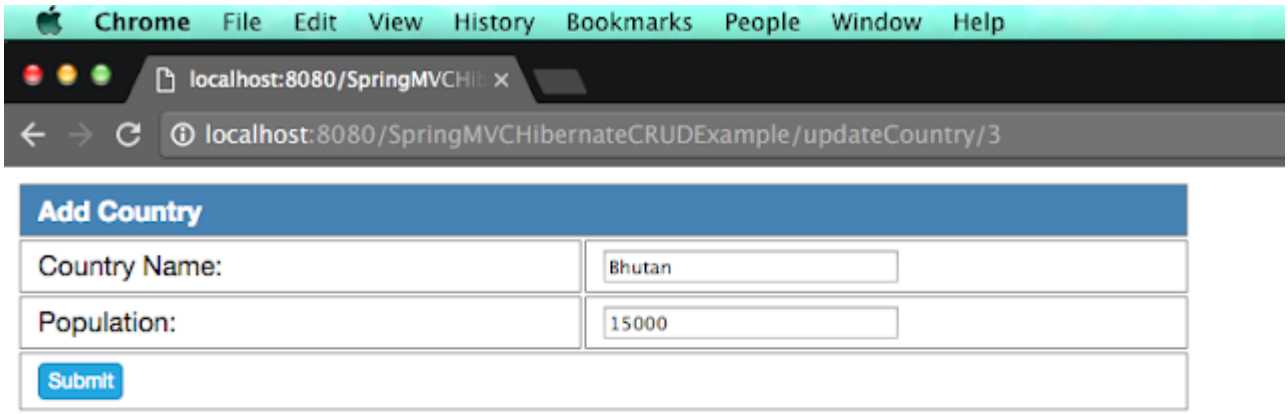
As you can see, we did not add any country to the list, so it is empty. Lets add Country named India to country list and click submit. Similarly we will add China , Bhutan and Nepal respectively and you will see below screen.



Country List

Id	Country Name	Population	Edit	Delete
1	India	40000	Edit	Delete
2	China	50000	Edit	Delete
3	Bhutan	20000	Edit	Delete
4	Nepal	30000	Edit	Delete

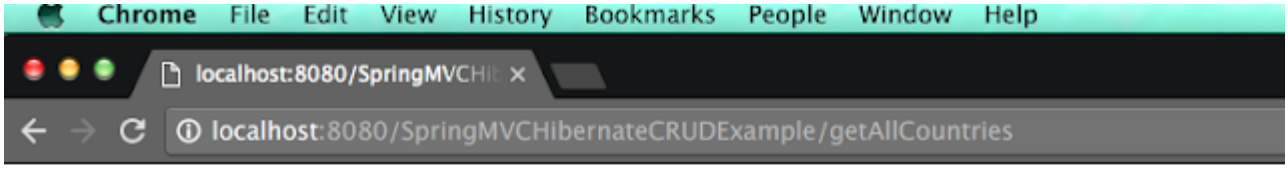
Lets edit population of Bhutan to 15000. Click on edit button corresponds to Bhutan.



Country List

Id	Country Name	Population	Edit	Delete
1	India	40000	Edit	Delete
2	China	50000	Edit	Delete
3	Bhutan	20000	Edit	Delete
4	Nepal	30000	Edit	Delete

When you click on submit, you will get below screen.

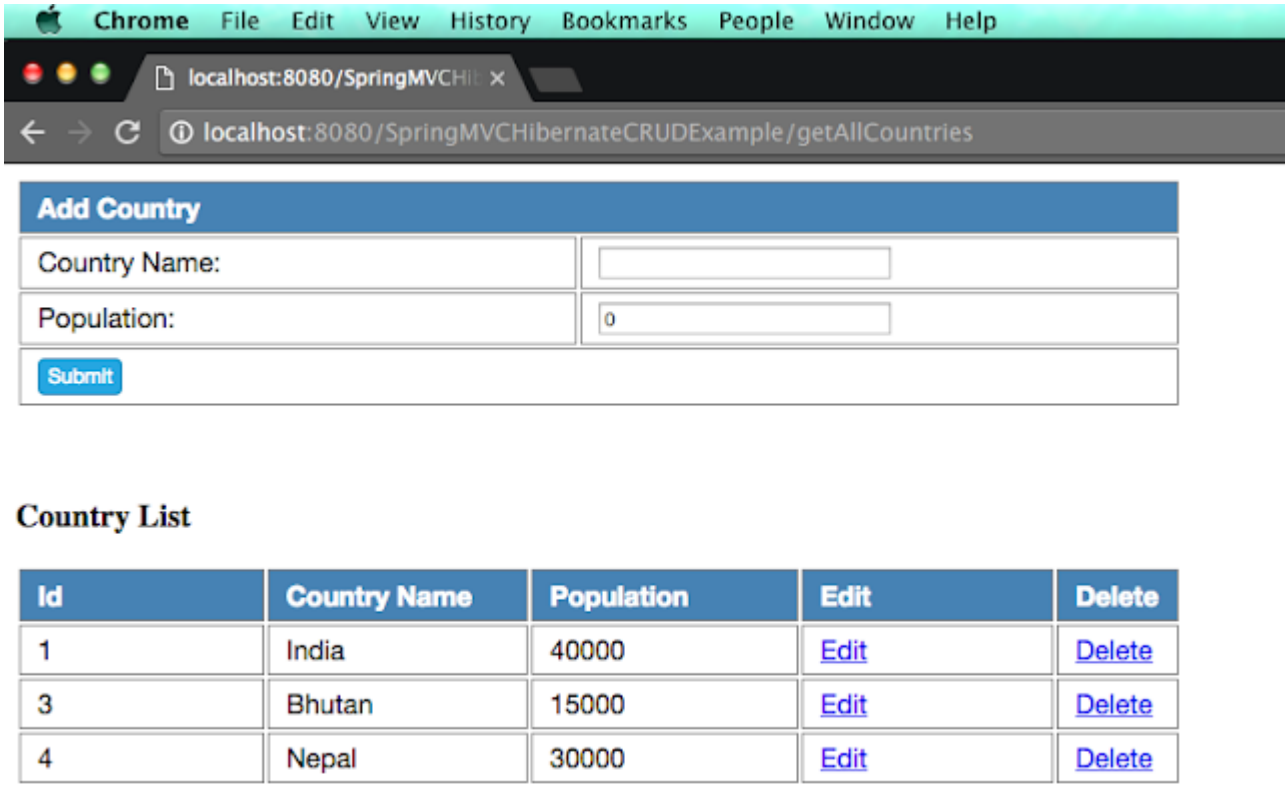


Add Country	
Country Name:	
Population:	0
Submit	

Country List

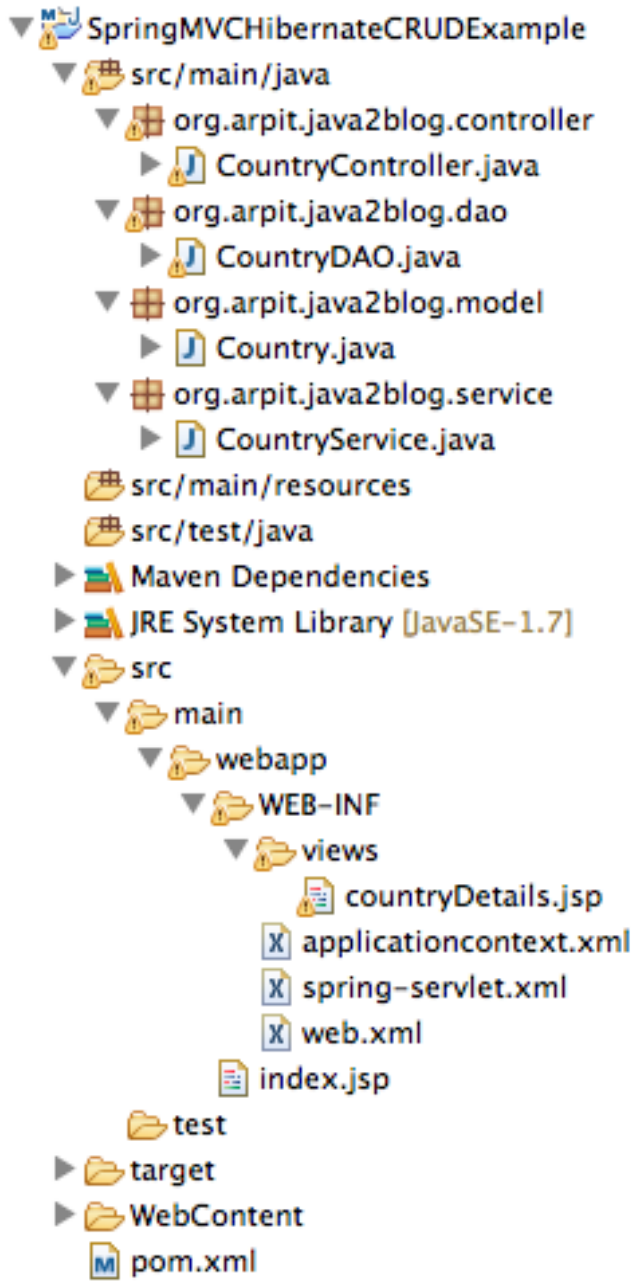
Id	Country Name	Population	Edit	Delete
1	India	40000	Edit	Delete
2	China	50000	Edit	Delete
3	Bhutan	15000	Edit	Delete
4	Nepal	30000	Edit	Delete

Lets delete country China from above list, click on delete button corresponds to 2nd row.
You will see below screen.



As you can see, china got deleted from above list.

Project structure:



We are done with Spring MVC hibernate MySQL CRUD example.

Spring Restful client – RestTemplate example::

We have already seen Spring restful web services crud example. We have used postmon utility to demonstrate all HTTP methods such as get, post, delete and put but if you want to write java code for restful client , you can use Spring RestTemplate.
You can always use java 's HttpClient but Spring RestTemplate provides more methods and options that you can use to consume Spring restful web services via Http methods.

Here is list of methods provided by Spring Resttemplate for each http methods.

Method	Spring RestTemplate's method
Get	getForObject, getForEntity
Post	postForObject(String url, Object reque responseType, String… uriVariables) postForLocation(String url, Object req String… urlVariables),
Put	put(String url, Object request, String… urlVariables)
Delete	delete()
Head	headForHeaders(String url, String… urlVariables)
Options	optionsForAllow(String url, String… urlVariables)

I am using same example which we have seen in Spring rest crud example. I am using java client instead of postman to consure Rest APIs.

Get example:

You can use getForObject or getForEntity for calling http get method.

Spring Rest API Code get method:

```
1
2  @RequestMapping(value = "/country/{id}", method = RequestMethod.GET, headers = "Accept=application/json")
3  public Country getCountryById(@PathVariable int id) {
4   return countryService.getCountry(id);
5  }
6
```

Spring RestTemplate get Method :

```
1
2  package org.arpit.java2blog.client;
3
4  import org.arpit.java2blog.bean.Country;
```

```
import org.springframework.web.client.RestTemplate;
/**
 * @author Arpit Mandliya
 */
public class SpringRestTemplateExample {

public static void main(String args[]) {

 RestTemplate restTemplate = new RestTemplate();
 Country bhutan = restTemplate
  .getForObject("http://localhost:8080/SpringRestfulWebServicesCRUDExample/country/{id}", Country.class,2
);
  System.out.println("Country Name:"+bhutan.getCountryName());
  System.out.println("Population:"+bhutan.getPopulation());
 }
}
```

When you run above code, you will get below output:

```
1
2 Country Name:Bhutan
3 Population:7000
4
```

Post example :

You can use PostForObject or PostForLocation to call post method.

Spring Rest API Code post method:

```
1
2 @RequestMapping(value = "/countries", method = RequestMethod.POST, headers = "Accept=application/json")
3 public Country addCountry(@RequestBody Country country) {
4  return countryService.addCountry(country);
5 }
6
```

Spring RestTemplate post Method :

```
1
```

```
package org.arpit.java2blog.client;

import org.arpit.java2blog.bean.Country;
import org.springframework.web.client.RestTemplate;
/**
 * @author Arpit Mandliya
 */
public class SpringRestTemplateExample {

 public static void main(String args[]) {

  RestTemplate restTemplate = new RestTemplate();

    final String uri = "http://localhost:8080/SpringRestfulWebServicesCRUDExample/countries";

    Country country = new Country();
    country.setCountryName("USA");
    country.setPopulation(4000);

    Country addedCountry = restTemplate.postForObject( uri, country, Country.class);
    System.out.println("Country added : " +addedCountry.getCountryName());
 }
}

```

When you run above code, you will get below output:

```

Country added : USA

```

Put example :

You can use put to call http put method.

Spring Rest API Code put method:

```

@RequestMapping(value = "/countries", method = RequestMethod.PUT, headers = "Accept=application/json")
```

```
3  public Country updateCountry(@RequestBody Country country) {
4   return countryService.updateCountry(country);
5  }
6
```

Spring RestTemplate put Method :

```
1
2  package org.arpit.java2blog.client;
3
4  import java.util.HashMap;
5  import java.util.Map;
6
7  import org.arpit.java2blog.bean.Country;
8  import org.springframework.web.client.RestTemplate;
9  /**
10  * @author Arpit Mandliya
11  */
12 public class SpringRestTemplateExample {
13
14  public static void main(String args[]) {
15
16      final String uriForPut = "http://localhost:8080/SpringRestfulWebServicesCRUDExample/countries";
17
18      Country country = new Country();
19      country.setId(2);
20      country.setCountryName("Bhutan");
21      country.setPopulation(10000);
22
23      RestTemplate restTemplate = new RestTemplate();
24      restTemplate.put ( uriForPut, country);
25  }
26 }
27
```

When you run above code, population for bhutan will be updated to 10000.

Delete example :

You can use delete to call http delete method.

Spring Rest API Code delete method:

```
1
2  @RequestMapping(value = "/country/{id}", method = RequestMethod.DELETE, headers = "Accept=application/js
   on")
3  public void deleteCountry(@PathVariable("id") int id) {
4   countryService.deleteCountry(id);
5
6  }
7
```

Spring RestTemplate delete Method :

```
1
2  package org.arpit.java2blog.client;
3
4  import java.util.HashMap;
5  import java.util.Map;
6
7  import org.springframework.web.client.RestTemplate;
8  /**
9   * @author Arpit Mandliya
10 */
11 public class SpringRestTemplateExample {
12
13  public static void main(String args[]) {
14
15   final String uriForDelete = "http://localhost:8080/SpringRestfulWebServicesCRUDExample/country/{id}";
16
17     Map<String, String> params = new HashMap<String, String>();
18     params.put("id","2");
19
20     RestTemplate restTemplate = new RestTemplate();
21     restTemplate.delete ( uriForDelete,params);
22 }
23 }
24
```

When you run above code, bhutan will be deleted from list of countries.

Android Restful web services:::

we are going to integrate android with restful web services which return json as response.
So we are going to get json from restful web services and then render json response to android custom listview.

Example :

I have already implemented restful webservices json example. I am going to use same example for implementation of restful web services.
If you are going to deploy above web service then use below url to fetch json data

```
1
2 <span style="color: green; font-weight: bold;">http://192.168.2.22:8080/JAXRSJsonExample/rest/countries</span>
3
```

Here **192.168.2.22** is IP address of my machine. Once you implement this web services on your machine, you should replace it with your ip address.
If you don't want to implement rest web service yourself and want below json response. You can use below link.
https://cdn.rawgit.com/arpitmandliya/AndroidRestJSONExample/master/countries.json
You will get below Json response from above URL:

```
1
2 [
3 {
4 "id":1,
5 "countryName":"India"
6 },
7 {
8 "id":4,
9 "countryName":"China"
10 },
11 {
12 "id":3,
13 "countryName":"Nepal"
14 },
15 {
16 "id":2,
```

```
17  "countryName":"Bhutan"
18  }
19 ]
20
```

We are getting above Json array from rest web service. If you are not familiar with JSON, you can go through Json tutorial. We are going to render above JSON response to android custom listview as below:

Lets code this example now:

Source code:

Download github source code

Step 1 :Creating Project

Create an android application project named “AndroidRestJsonExample”.

Step 2 : Creating Layout

Change res ->layout -> activity_main.xml as below:

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:id="@+id/activity_main"
6   android:layout_width="match_parent"
7   android:layout_height="match_parent"
8   android:paddingBottom="@dimen/activity_vertical_margin"
9   android:paddingLeft="@dimen/activity_horizontal_margin"
10  android:paddingRight="@dimen/activity_horizontal_margin"
11  android:paddingTop="10dp"
12  tools:context="com.java2blog.androidrestjsonexample.MainActivity">
13
14  <Button
15  android:layout_width="match_parent"
16  android:layout_height="wrap_content"
17  android:text="Fetching countries data"
18  android:id="@+id/btnSubmit"
19  />
```

```
20
21 <ListView
22 android:id="@+id/android:list"
23 android:layout_width="match_parent"
24 android:layout_height="wrap_content"
25 android:layout_below="@id/btnSubmit"
26 />
27 </RelativeLayout>
28
```

We have one button and listview. When you click the button, we are going to populate data in listview by calling restful web services and render it on the listview.

Step 3: Creating layout for Row

As We have declared ListView widget in activity_main.xml. Now we need to provide layout for individual row.

- Go to res -> layout
- right click on layout
- Click on New -> File.
- Create a file named “row_item.xml” and paste below code in row_item.xml.

```
1
2 <?xml version="1.0" encoding="utf-8"?>
3 <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
4 android:layout_width="match_parent"
5 android:layout_height="match_parent" >
6 <TextView
7 android:layout_width="wrap_content"
8 android:layout_height="wrap_content"
9 android:layout_columnWeight="1"
10 android:layout_marginLeft="10dp"
11 android:textSize="30dp"
12 android:textColor="#1E90FF"
13 android:id="@+id/textViewId"
14 android:layout_row="0"
15 android:layout_column="1" />
16 <TextView
```

```
17  android:layout_width="wrap_content"
18  android:layout_height="wrap_content"
19  android:layout_columnWeight="1"
20  android:layout_marginLeft="10dp"
21  android:textSize="20dp"
22  android:textColor="#4B0082"
23  android:id="@+id/textViewCountry"
24  android:layout_row="1"
25  android:layout_column="1" />
26 </GridLayout>
27
```

Step 4 : Creating model object Country :

If you notice, we are getting Json array from restful web service which has 4 items. Each item have two attributes id and countryName. We are going to create country object for each item.

```
 1
 2  package com.java2blog.androidrestjsonexample;
 3  public class Country{
 4     int id;
 5     String countryName;
 6
 7     public Country(int i, String countryName) {
 8        super();
 9        this.id = i;
10        this.countryName = countryName;
11     }
12     public int getId() {
13        return id;
14     }
15     public void setId(int id) {
16        this.id = id;
17     }
18     public String getCountryName() {
19        return countryName;
20     }
```

```
21    public void setCountryName(String countryName) {
22        this.countryName = countryName;
23    }
24 }
25
```

Step 5 : Creating BaseAdapter for ListView

Before creating MainActivity, we need to create CustomCountryList class for custom ListView row.

```
 1
 2  package com.java2blog.androidrestjsonexample;
 3
 4  import android.app.Activity;
 5  import android.view.LayoutInflater;
 6  import android.view.View;
 7  import android.view.ViewGroup;
 8  import android.widget.BaseAdapter;
 9  import android.widget.TextView;
10
11  import com.java2blog.androidrestjsonexample.Country;
12  import com.java2blog.androidrestjsonexample.R;
13
14  import java.util.ArrayList;
15
16  public class CustomCountryList extends BaseAdapter {
17
18
19     private Activity context;
20     ArrayList countries;
21
22
23     public CustomCountryList(Activity context, ArrayList countries) {
24      //   super(context, R.layout.row_item, countries);
25        this.context = context;
26        this.countries=countries;
27
```

```
28  }
29
30  public static class ViewHolder
31  {
32     TextView textViewId;
33     TextView textViewCountry;
34  }
35  @Override
36  public View getView(int position, View convertView, ViewGroup parent) {
37     View row=convertView;
38
39     LayoutInflater inflater = context.getLayoutInflater();
40     ViewHolder vh;
41     if(convertView==null) {
42        vh=new ViewHolder();
43        row = inflater.inflate(R.layout.row_item, null, true);
44        vh.textViewId = (TextView) row.findViewById(R.id.textViewId);
45        vh.textViewCountry = (TextView) row.findViewById(R.id.textViewCountry);
46        // store the holder with the view.
47        row.setTag(vh);
48     }
49     else {
50        vh = (ViewHolder) convertView.getTag();
51     }
52
53     vh.textViewCountry.setText(countries.get(position).getCountryName());
54     vh.textViewId.setText(""+countries.get(position).getId());
55
56     return  row;
57  }
58
59  public long getItemId(int position) {
60     return position;
61  }
62
```

```
63    public Object getItem(int position) {
64        return position;
65    }
66
67    public int getCount() {
68
69        if(countries.size()<=0)
70            return 1;
71        return countries.size();
72    }
73 }
74
```

This class is used to populating data for ListVIew. getView method is get called for drawing each row.

Step 6 : Creating MainActivity

We are going to create a addition inner class GetServerData which extends AsyncTask. This class will make https call in background thread. Here are three methods which you may need to implement while using AsyncTask.

- **onPreExecute() :** This method will get called before making HTTP call. We are initializing ProgressDialog here.
- **doInBackground :** We are going to make HTTP call in this method. We have use HttpURLConnection class to make HTTP call. You can not access any UI elements while executing this method
- **onPostExecute :** This executes after execution of doInBackground method. In this method, we are going to create CustomCountryList object and populate listview data.

Change src/main/packageName/MainActivity.java as below:

```
1
2   package com.java2blog.androidrestjsonexample;
3
4   import android.app.Activity;
5   import android.app.ProgressDialog;
6   import android.os.AsyncTask;
7   import android.os.Bundle;
8   import android.os.StrictMode;
9   import android.support.v7.app.AppCompatActivity;
10  import android.util.Log;
```

```
11 import android.view.View;
12 import android.widget.AdapterView;
13 import android.widget.Button;
14 import android.widget.ListView;
15 import android.widget.Toast;
16 import org.json.JSONArray;
17 import org.json.JSONException;
18 import org.json.JSONObject;
19 import java.io.BufferedReader;
20 import java.io.InputStreamReader;
21 import java.net.HttpURLConnection;
22 import java.net.URL;
23 import java.util.ArrayList;
24
25 import javax.net.ssl.HttpsURLConnection;
26
27 import static android.content.ContentValues.TAG;
28
29 public class MainActivity extends AppCompatActivity {
30
31     private Button btnSubmit;
32     String responseText;
33     StringBuffer response;
34     URL url;
35     Activity activity;
36     ArrayList countries=new ArrayList();
37     private ProgressDialog progressDialog;
38     ListView listView;
39     // In case if you deploy rest web service, then use below link and replace below ip address with yours
40     //http://192.168.2.22:8080/JAXRSJsonExample/rest/countries
41
42     //Direct Web services URL
43     private String path = "https://cdn.rawgit.com/arpitmandliya/AndroidRestJSONExample/master/countries.js
   on";
44
45
       @Override
```

```
46  protected void onCreate(Bundle savedInstanceState) {
47      super.onCreate(savedInstanceState);
48      setContentView(R.layout.activity_main);
49      activity = this;
50      btnSubmit = (Button) findViewById(R.id.btnSubmit);
51      listView = (ListView) findViewById(android.R.id.list);
52      btnSubmit.setOnClickListener(new View.OnClickListener() {
53          @Override
54          public void onClick(View v) {
55              countries.clear();
56              //Call WebService
57              new GetServerData().execute();
58          }
59      });
60  }
61
62  class GetServerData extends AsyncTask
63  {
64
65      @Override
66      protected void onPreExecute() {
67          super.onPreExecute();
68          // Showing progress dialog
69          progressDialog = new ProgressDialog(MainActivity.this);
70          progressDialog.setMessage("Fetching conntry data");
71          progressDialog.setCancelable(false);
72          progressDialog.show();
73
74      }
75
76      @Override
77      protected Object doInBackground(Object[] objects) {
78          return getWebServiceResponseData();
79      }
80
```

```
81          @Override
82          protected void onPostExecute(Object o) {
83              super.onPostExecute(o);
84
85              // Dismiss the progress dialog
86              if (progressDialog.isShowing())
87                  progressDialog.dismiss();
88              // For populating list data
89              CustomCountryList customCountryList = new CustomCountryList(activity, countries);
90              listView.setAdapter(customCountryList);
91
92              listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
93                  @Override
94                  public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {
95                      Toast.makeText(getApplicationContext(),"You Selected "+countries.get(position).getCountryName
    ()+ " as Country",Toast.LENGTH_SHORT).show();       }
96              });
97          }
98      }
99      protected Void getWebServiceResponseData() {
100
101         try {
102
103             url = new URL(path);
104             Log.d(TAG, "ServerData: " + path);
105             HttpURLConnection conn = (HttpURLConnection) url.openConnection();
106             conn.setReadTimeout(15000);
107             conn.setConnectTimeout(15000);
108             conn.setRequestMethod("GET");
109
110             int responseCode = conn.getResponseCode();
111
112             Log.d(TAG, "Response code: " + responseCode);
113             if (responseCode == HttpsURLConnection.HTTP_OK) {
114                 // Reading response from input Stream
115                 BufferedReader in = new BufferedReader(
```

```
116                     new InputStreamReader(conn.getInputStream()));
117                 String output;
118                 response = new StringBuffer();
119
120                 while ((output = in.readLine()) != null) {
121                     response.append(output);
122                 }
123                 in.close();
124               }}
125             catch(Exception e){
126                 e.printStackTrace();
127             }
128
129               responseText = response.toString();
130               //Call ServerData() method to call webservice and store result in response
131              //  response = service.ServerData(path, postDataParams);
132               Log.d(TAG, "data:" + responseText);
133               try {
134                 JSONArray jsonarray = new JSONArray(responseText);
135                 for (int i = 0; i < jsonarray.length(); i++) {
136                     JSONObject jsonobject = jsonarray.getJSONObject(i);
137                     int id = jsonobject.getInt("id");
138                     String country = jsonobject.getString("countryName");
139                     Log.d(TAG, "id:" + id);
140                     Log.d(TAG, "country:" + country);
141                     Country countryObj=new Country(id,country);
142                     countries.add(countryObj);
143                 }
144
145               } catch (JSONException e) {
146                 e.printStackTrace();
147               }
148               return null;
149             }
150     }
```

151

Step 7: Add internet permission in AndroidManifest.xml

Copy following code:

```
1
2 <uses-permission android:name="android.permission.INTERNET" />
3
```

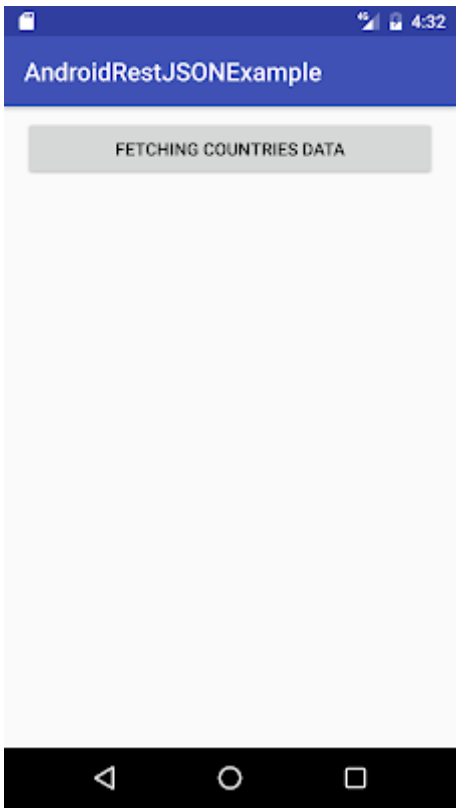
Put it in AndroidManifest.xml

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4   package="com.java2blog.androidrestjsonexample">
5
6   <uses-permission android:name="android.permission.INTERNET" />
7   <application
8   android:allowBackup="true"
9   android:icon="@mipmap/ic_launcher"
10  android:label="@string/app_name"
11  android:supportsRtl="true"
12  android:theme="@style/AppTheme">
13  <activity android:name=".MainActivity">
14  <intent-filter>
15  <action android:name="android.intent.action.MAIN" />
16
17  <category android:name="android.intent.category.LAUNCHER" />
18  </intent-filter>
19  </activity>
20  </application>
21
22 </manifest>
23
```

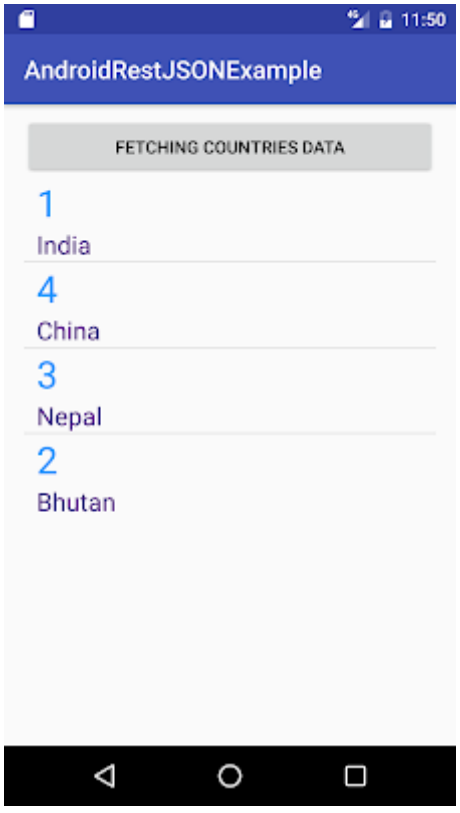
Done, we have added internet permission to AndroidManifest.xml. Your application must be able to access internet now.

Step 8 : Running the app

When you run the app, you will get below screen:



When you click on above button, you will get below screen.



We are done with Android Restful web services json example

Top 20 Web services interview questions::

Web services interview questions are most asked questions if you are applying for software developer role.
In this post, we will see multiple web services interview questions.

1. What are web services?

Web services are ways of communication between two application over network. It allows you to expose business logic using API.

For example:
Lets say you are java developer, you can create web service and expose API over internet and any other developer (lets say .net developer) can access it.

2. What are features of web services?

- Interoperability
- Reuse already developed(old) functionality into new software:
- Loosely Coupled
- Extensibility

3. What are different types of web services?

- SOAP
- Restful web services

4. What is SOAP?

SOAP stands for Simple object access protocol. It is protocol to exchange information using request and response in XML format over transport protocol such as HTTP, SMTP etc.

5. What are important components for SOAP?

- Simple access object protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration(UDDI)

6. What is WSDL?

WSDL stands for Web Service Description Language. It is an XML file that describes the technical details of how to implement a web service, more specifically the URI, port, method names, arguments, and data types. You can understand following details using WSDL

- Port / Endpoint – URL of the web service
- Input message format
- Output message format
- Security protocol that needs to be followed
- Which protocol the web service uses

7. What is UDDI?

UDDI stands for Universal Description, Discovery and Integration.It is a directory service. Web service provider can register themselves with a UDDI and make themselves available through it for discovery.

8. What is JAX-WS?

JAX-WS stands for Java API for XML Web Services. JAX-WS is standard XML based Java API which is used to create SOAP web services.

9. What are some important annotations for JAX-WS?

- @WebService
- @WebMethod
- @SOAPBinding

10. What do you mean by end point in terms of SOAP?

End point is nothing but URL which other application can use to access it.

for example:

end point:http://localhost:8080/WS/HelloWorld

11. How can you access WSDL for web service?

You just need to put ?wsdl at the end of end point URL.

for example:

end point:http://localhost:8080/WS/HelloWorld
WSDL url: http://localhost:8080/WS/HelloWorld?wsdl

12. What is wsimport?

wsimport is utility which generates java classes from WSDL. It is part of JDK 6.

13.What is sun-jaxws.xml file?

This file provides endpoint details about JAX-WS web service which is deployed on tomcat.It is available at WEB-INF directory.
For example:

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
4   <endpoint
5     name="HelloWorldWS"
6     implementation="org.arpit.javapostsforlearning.webservice.HelloWorldImpl"
7     url-pattern="/HelloWorldWS"/>
8 </endpoints>
9
```

14. What are Restful web services?

In the web services terms, REpresentational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web services client uses that URI to access the resource.

15.What are HTTP methods that can be used with Restful web services?

Mainly used HTTP methods are GET, POST, PUT, DELETE, HEAD and OPTIONS

16. What is JAX-RS?

Java API for RESTful Web Services (**JAX-RS**), is a set if APIs to create web service which supports REST architecture. JAX-RS is part of the Java EE6, and help developers to create REST web application easily.

17. What are some important annotations which you use to create Restful web services?

Some of important annotations which are used for creating web services are:

@Path : This is used to set path for URI at class level or method level
@GET,@POST,@PUT,@DELETE : There are annotations corresponds to HTTP methods
@Produces(MediaType.TEXT_XML [, more-types]): @Produces defines which MIME type is delivered by a method
@PathParam: Used to inject values from the URL into a method parameter.
@Consumes(**MediaType.TEXT_XML**) : @Cosumes defines which MIME type will be consumed by the method .

18. What are ways to test SOAP web services?

For testing SOAP :
SOAPUI
For testing Restful web services:

- Postman for chrome browser
- poster for firefox

19. How to choose between REST and SOAP web services?

- If you want to implement web services in less time, go with REST
- If you know your client beforehand , then you can choose SOAP. If you are not aware about clients then go with REST.
- If you want to work with different format other than XML, go with REST. SOAP only supports XML format.

20. What are differences between SOAP and REST web services?

Refer above ….

Restful web services interview questions::

Last updated : March 28th, 2017

NO COMMENTS

Restful web services interview questions

Previous

Next

Restful web services are very popular now a days because it is very simple to implement and less time consuming. In this post, we are going to see restful web services interview questions with answers.

1. What is REST?

REST is an architectural style which was brought in by Roy Fielding in 2000 in his doctoral thesis.

In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.

2. What are Restful web services?

In the web services terms, REpresentational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web services client uses that URI to access the resource.
It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.

3. What are important features of Restful web services?

Some important features of Restful web services are:
Resource identification through URI:Resources are identified by their URIs (typically links on internet). So, a client can directly access a RESTful Web Services using the URIs of the resources (same as you put a website address in the browser's address bar and get some representation as response).
Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.
Client-Server: A clear separation concerns is the reason behind this constraint. Separating concerns between the Client and Server helps improve portability in the Client and Scalability of the server components.
Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
Cache: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
Named resources – the system is comprised of resources which are named using a URL.
Interconnected resource representations – the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.
Layered components – intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.
Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

4. What are HTTP methods that can be used in Restful web services?

RESTful web services use HTTP protocol methods for the operations they perform.
Some important Methods are:
GET : It defines a reading access of the resource without side-effects.This operation is idempotent i.e.they can be applied multiple times without changing the result
PUT : It is generally used for updating resource. It must also be idempotent.
DELETE : It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
POST : It is used for creating a new resource. It is not idempotent.

5. What do you mean by Idempotent and which HTTP methods are idempotent?

Idempotent means result of multiple successful request will not change state of resource after initial application

For example : Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted. Get, put and delete are HTTP Idempotent methods.

6. What are differences between Post and Put Http methods?

POST : It is used for creating a new resource. It is not idempotent.
PUT : It is generally used for updating resource. It is idempotent. Idempotent means result of multiple successful request will not change state of resource after initial application

7. What happens if resources are shared by multiple clients? Do you need to make it thread safe explicitly?

New resource instance is created for each request, so you don't need to implement thread safety or synchronization aid. It is by default thread safe.

8. What is JAX-RS?

Java API for RESTful Web Services (**JAX-RS**), is a set if APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.

9. What are REST frameworks that you are aware of and which can be used to create Restful webservices?

There are multiple Rest framework that can be used to create Restful web services such as

- Jersey
- RestEasy
- Restlet
- CFX
- Spring Rest webservices

10. What are some important annotations which you use to create Restful web services?

Some of important annotations which are used for creating web services are:

@Path : This is used to set path for URI at class level or method level
@GET,@POST,@PUT,@DELETE : There are annotations corresponds to HTTP methods
@Produces(MediaType.TEXT_XML [, more-types]): @Produces defines which MIME type is delivered by a method
@PathParam: Used to inject values from the URL into a method parameter.
@Consumes(**MediaType.TEXT_XML**) : @Cosumes defines which MIME type will be consumed by the method .

11. Can you use get method to create Resources rather than post?

No, Get should be used only for resource retrieval and not for resource creation.

12. What are ways to test Restful web services?

You require a restful client to test restful web services. You can use:

- Postman for chrome browser
- poster for firefox