Business Process Digitalization and Cloud Computing

4. Architecture fundamentals

Andrea Morichetta, Phd

Computer Science Division

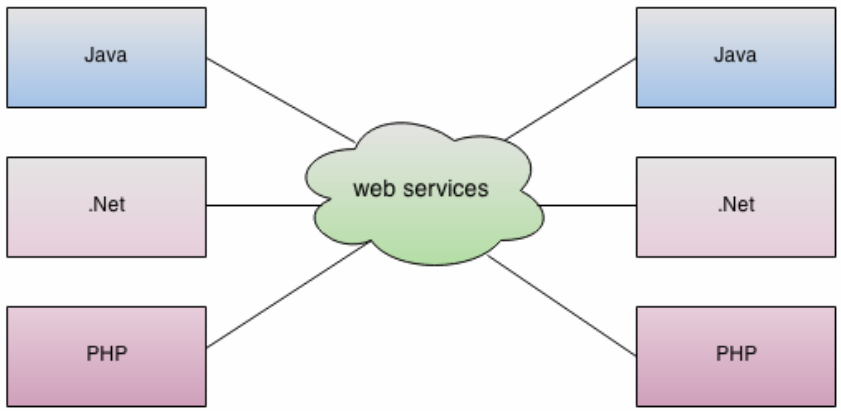
October 30, 2018

Table of contents

What is Web Service?

A Web Service can be defined by following ways:

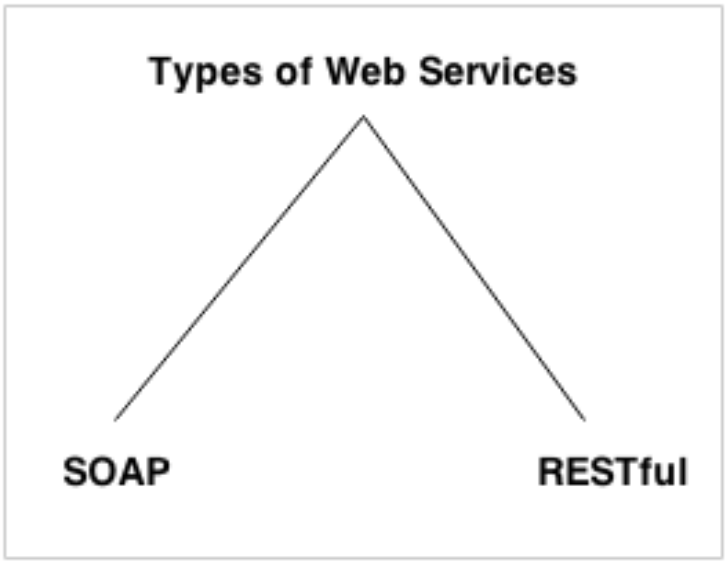
- is a **client server application** or **application component** for communication.
- **method of communication** between two devices over network.
- is a software system for **interoperable machine to machine communication**.
- is a **collection of standards or protocols** for exchanging information between two devices or application.



Type of Web Service

There are mainly two types of web services.

- SOAP web services.
- RESTful web services.



SOAP Web Services

SOAP stands for Simple Object Access Protocol. It is a **XML-based protocol** for accessing web services.

SOAP is a **W3C recommendation for communication** between two applications.

SOAP is **XML based protocol**. It is platform independent and language independent.

Advantages of Soap Web Services

WS Security: SOAP defines its own security known as WS Security.

Language and Platform independent: SOAP web services can be written in any programming language and executed in any platform.

Disadvantages of Soap Web Services

Slow: SOAP uses XML format that must be parsed to be read. So it is slow and consumes more bandwidth and resource.

WSDL dependent: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

RESTful Web Services

REST stands for **REpresentational State Transfer**.
REST is an **architectural style not a protocol**.

Advantages of RESTful Web Services

Fast: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
Language and Platform independent: RESTful web services can be written in any programming language and executed in any platform.
Can use SOAP: RESTful web services can use SOAP web services as the implementation.
Permits different data format: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

SOAP vs REST

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.

Service Oriented Architecture (SOA)

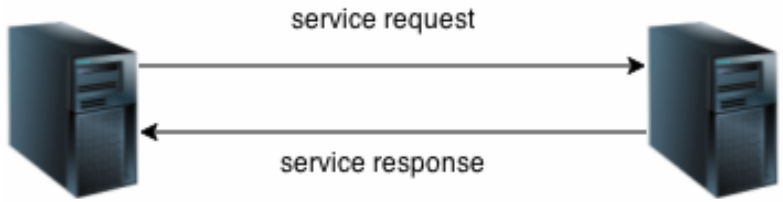
Service Oriented Architecture or SOA is a **design pattern**. It is designed to provide services to other applications through protocol. It **is a concept only** and not tied to any programming language or platform.

Service

A service is **well-defined**, **self-contained** function that represents unit of functionality. A service can exchange information from another service. It is not dependent on the state of another service.

Service Connections

Service consumer sends service request to the service provider and service provider sends the service response to the service consumer. **The service connection is understandable to both service consumer and service provider**.



Java Web Services API

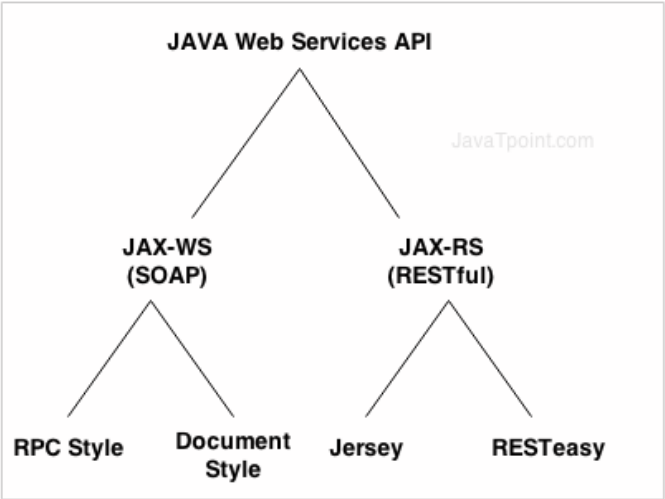
There are two ways to write java web service application code: SOAP and RESTful. There are two main API's defined by Java for developing web service applications.

JAX-WS

for SOAP web services. The are two ways to write JAX-WS application code: by RPC style and Document style.

JAX-RS

for RESTful web services. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.



Restful Web Services

Restful Web Services is a **stateless client-server architecture** where web services are resources and can be **identified by their URIs**.

REST Client applications can use **HTTP GET/POST methods** to invoke Restful web services. REST doesn't specify any specific protocol to use, but in almost all cases it's used over HTTP/HTTPS.

When compared to SOAP web services, these are lightweight and doesn't follow any standard. We can use XML, JSON, text or any other type of data for request and response.

Restful Web Services Annotations

Some of the important JAX-RS **annotations** are:

- **@Path**: used to specify the relative path of class and methods. We can get the URI of a webservice by scanning the Path annotation value.
- **@GET, @PUT, @POST, @DELETE** and **@HEAD**: used to specify the HTTP request type for a method.
- **@Produces, @Consumes**: used to specify the request and response types.
- **@PathParam**: used to bind the method parameter to path value by parsing it.

Restful Web Services and SOAP

- SOAP is a **protocol** whereas REST is an **architectural style.**
- **SOAP server and client applications are tightly coupled** and bind with the WSDL contract whereas there is **no contract in REST web services and client**.
- **Learning curve is easy for REST** when compared to SOAP web services.
- REST web services request and response types can be XML, JSON, text etc. whereas SOAP works with XML only.
- JAX-RS is the Java API for REST web services whereas JAX-WS is the Java API for SOAP web services.

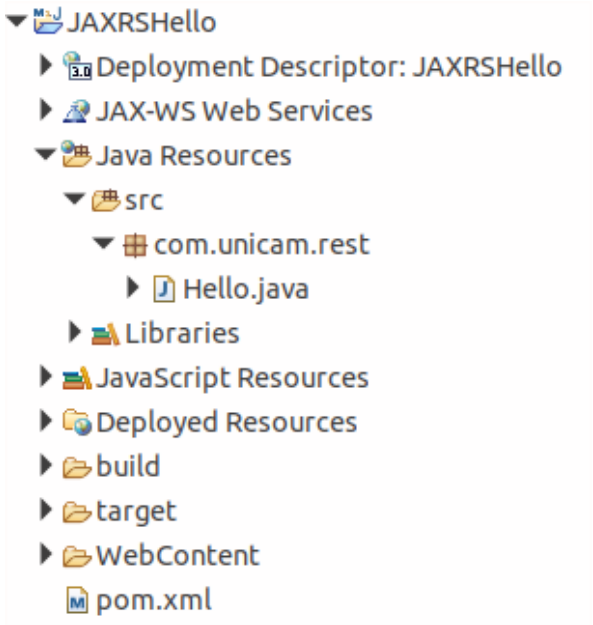
REST API Implementations

There are two major implementations of JAX-RS API: **Jersey** and **RESTEasy**.

Jersey: Jersey is the reference implementation provided by Sun.
For using Jersey as our JAX-RS implementation, all we need to configure its **servlet** in **web.xml** and add **required dependencies**. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.

Jersey Restful Web Services

(Step 1) Create a **dynamic web project** (right click one the project$-$ $>$ configure $-$ $>$ convert ..) and then **convert it to Maven** to get the skeleton of your web services project.



Jersey Restful Web Services

(Step 2) Let's look at the Jersey dependencies we have in pom.xml file.

```
            </plugins>
        </build>
        <dependencies>
            <dependency>
                <groupId>org.glassfish.jersey.core</groupId>
                <artifactId>jersey-common</artifactId>
                <version>2.27</version>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.containers</groupId>
                <artifactId>jersey-container-servlet</artifactId>
                <version>2.27</version>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.inject</groupId>
                <artifactId>jersey-hk2</artifactId>
                <version>2.27</version>
            </dependency>
        </dependencies>
</project>
```

Jersey Restful Web Services

(Step 3) Let's look at the deployment descriptor to learn how to configure Jersey to create our web application.

WebContent/WEB-INF/web.xml

```
<?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"
id="WebApp_ID" version="3.0">
 <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.unicam.rest</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

(Step 4) Implement the java class

```java
package com.unicam.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class Hello {
  // This method is called if HTML and XML is not requested
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "Hello Jersey Plain ";
  }
  // This method is called if XML is requested
  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey " + "</hello>";
  }

  // This method is called if HTML is requested
  @GET
  @Produces(MediaType.TEXT_HTML)
  public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey HTML" + "</h1></body>" + "</html> ";
  }
}
```

(Step 5) Implement the index.html

```
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<a href="rest/hello">Click Here</a>
</body>
</html>
```

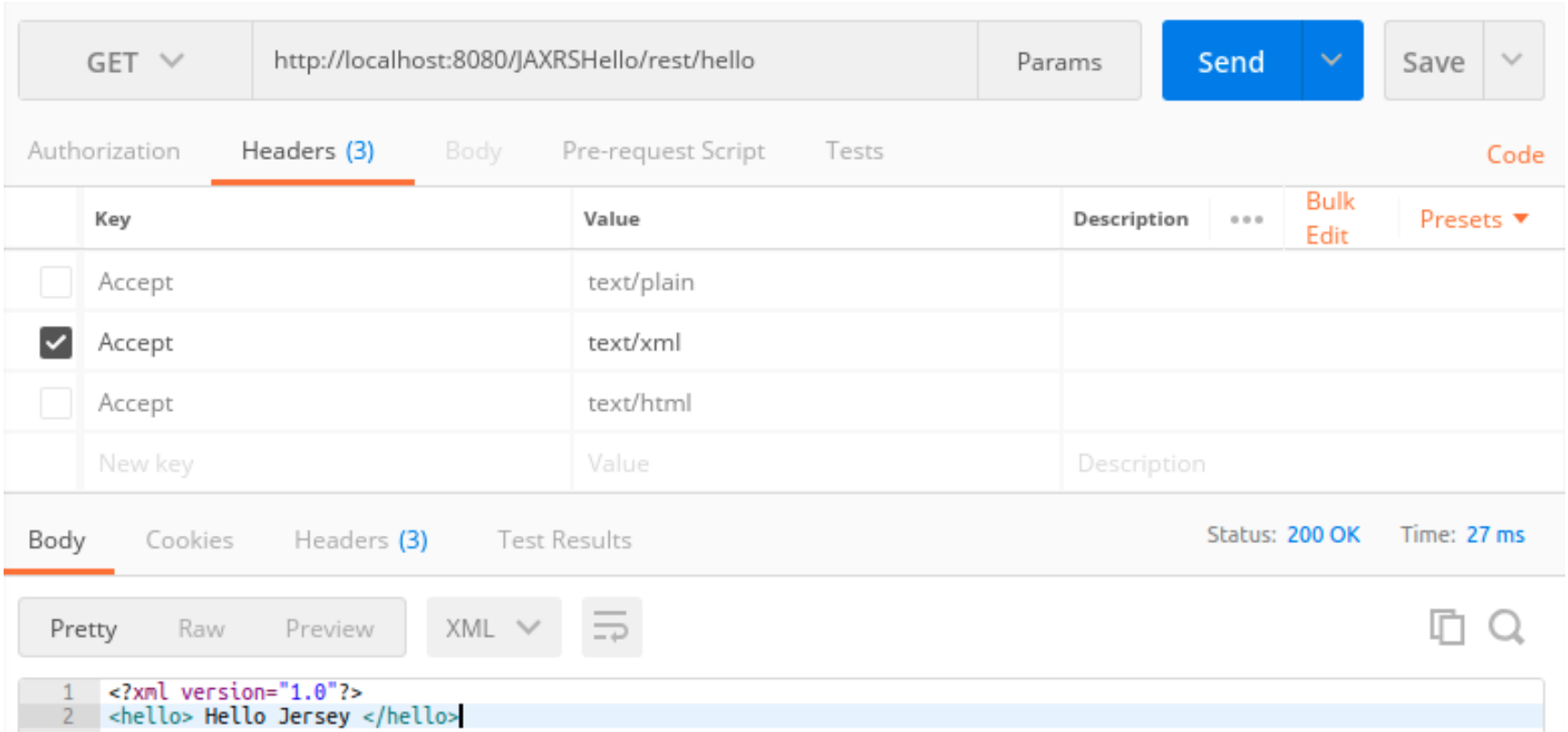
URL site:

`http://localhost:8080/JAXRSHello/`

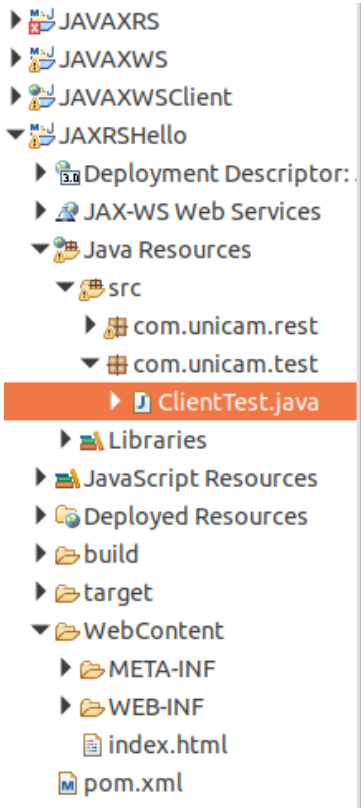
URL Web Service:

`http://localhost:8080/JAXRSHello/rest/hello`

(Step 6) Test with Postman

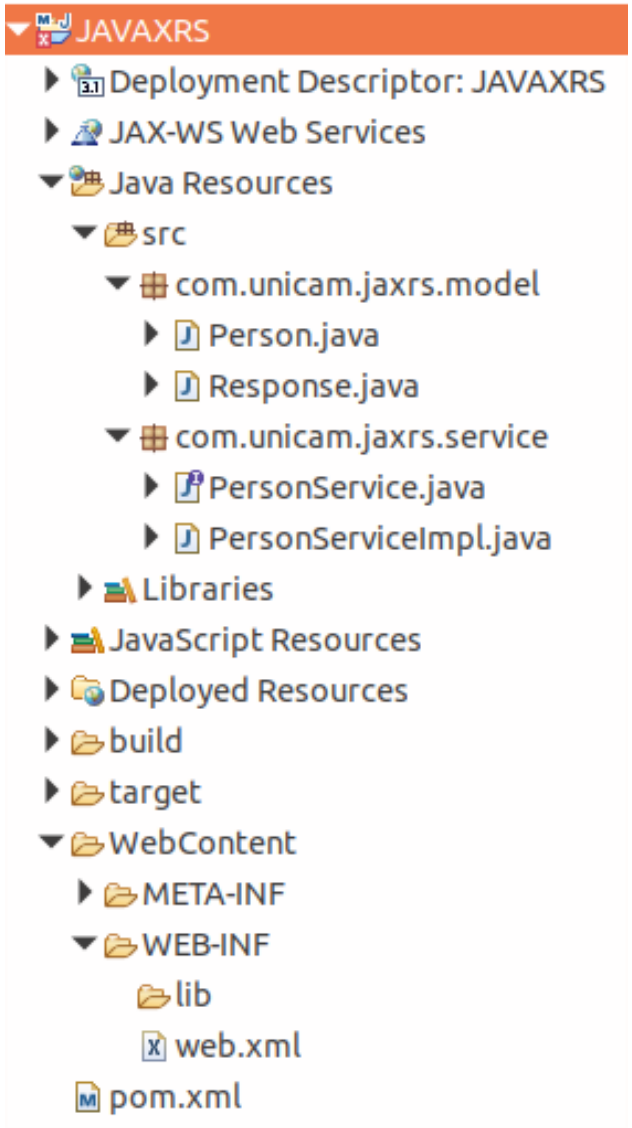


Client Test Hello




```java
import java.net.URI;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;
import org.glassfish.jersey.client.ClientConfig;
public class ClientTest {
  public static void main(String[] args) {
    ClientConfig config = new ClientConfig();
    Client client = ClientBuilder.newClient(config);
    WebTarget target = client.target(getBaseURI());
    //Now printing the server code of different media type
    System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_PLAIN).get(String.class));
    System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_XML).get(String.class));
    System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_HTML).get(String.class));
  }
  private static URI getBaseURI() {
    //here server is running on 4444 port number and project name is restfuljersey
    return UriBuilder.fromUri("http://localhost:8080/JAXRSHello/").build();
  }
}
```

More Complex Project



Java Restful Web Services Tutorial

URI	HTTP METHOD	DESCRIPTION
/person/{id}/getDummy	GET	Returns a dummy person object
/person/add	POST	Adds a person
/person/{id}/delete	GET	Delete the person with 'id' in the URI
/person/getAll	GET	Get all persons
/person/{id}/get	GET	Get the person with 'id' in the URI

Person.java

```java
package com.unicam.jaxrs.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement (name="person")
public class Person {
    private String name;
    private int age;
    private int id;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Override
    public String toString(){
        return id+"::"+name+"::"+age;
    }
```

Response.java

```java
package com.unicam.jaxrs.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Response {

    private boolean status;
    private String message;

    public boolean isStatus() {
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

PersonService.java

```java
package com.unicam.jaxrs.service;

import com.unicam.jaxrs.model.Person;

public interface PersonService {

    public Response addPerson(Person p);

    public Response deletePerson(int id);

    public Person getPerson(int id);

    public Person[] getAllPersons();

}
```

PersonServiceImpl.java

```java
package com.unicam.jaxrs.service;


import java.util.HashMap;

@Path("/person")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public class PersonServiceImpl implements PersonService {

    private static Map<Integer,Person> persons = new HashMap<Integer,Person>();

    @Override
    @POST
    @Path("/add")
    public Response addPerson(Person p) {
        Response response = new Response();
        if(persons.get(p.getId()) != null){
            response.setStatus(false);
            response.setMessage("Person Already Exists");
            return response;
        }
        persons.put(p.getId(), p);
        response.setStatus(true);
        response.setMessage("Person created successfully");
        return response;
    }
```

```java
@Override
@GET
@Path("/{id}/delete")
public Response deletePerson(@PathParam("id") int id) {
    Response response = new Response();
    if(persons.get(id) == null){
        response.setStatus(false);
        response.setMessage("Person Doesn't Exists");
        return response;
    }
    persons.remove(id);
    response.setStatus(true);
    response.setMessage("Person deleted successfully");
    return response;
}

@Override
@GET
@Path("/{id}/get")
public Person getPerson(@PathParam("id") int id) {
    return persons.get(id);
}

@GET
@Path("/{id}/getDummy")
public Person getDummyPerson(@PathParam("id") int id) {
    Person p = new Person();
    p.setAge(99);
    p.setName("Dummy");
    p.setId(id);
    return p;
}
```

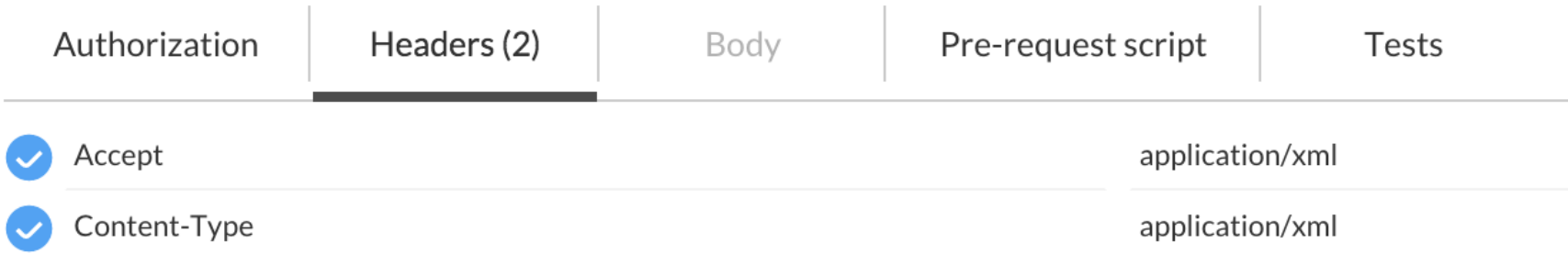
```java
@Override
@GET
@Path("/getAll")
public Person[] getAllPersons() {
    Set<Integer> ids = persons.keySet();
    Person[] p = new Person[ids.size()];
    int i=0;
    for(Integer id : ids){
        p[i] = persons.get(id);
        i++;
    }
    return p;
}
```

web.xml

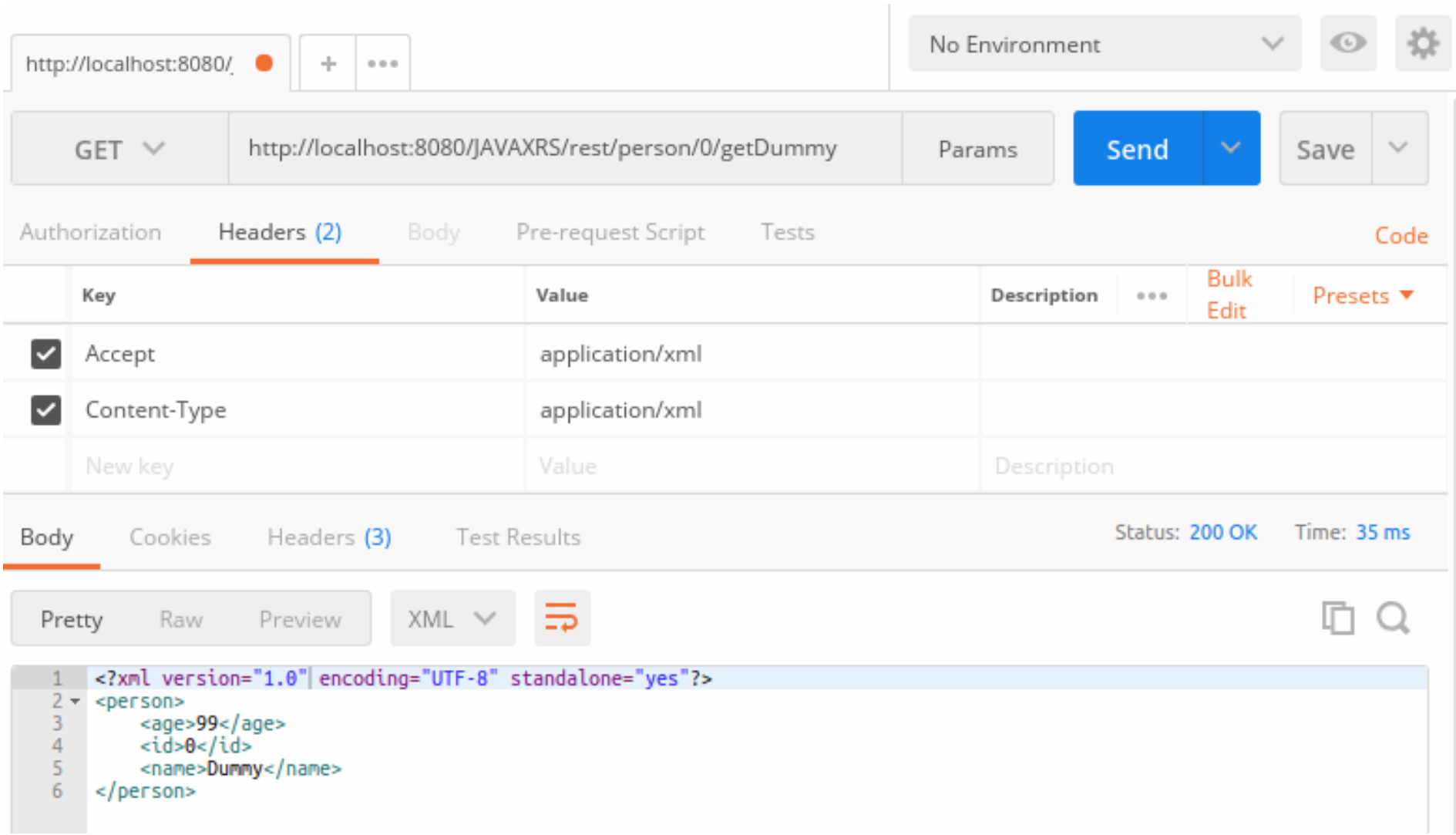
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns
  <display-name>JAXRS-Example</display-name>

<!-- Jersey Servlet configurations -->
    <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>com.unicam.jaxrs.service</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
  <!-- Jersey Servlet configurations -->

</web-app>
```

Restful Web Services Test

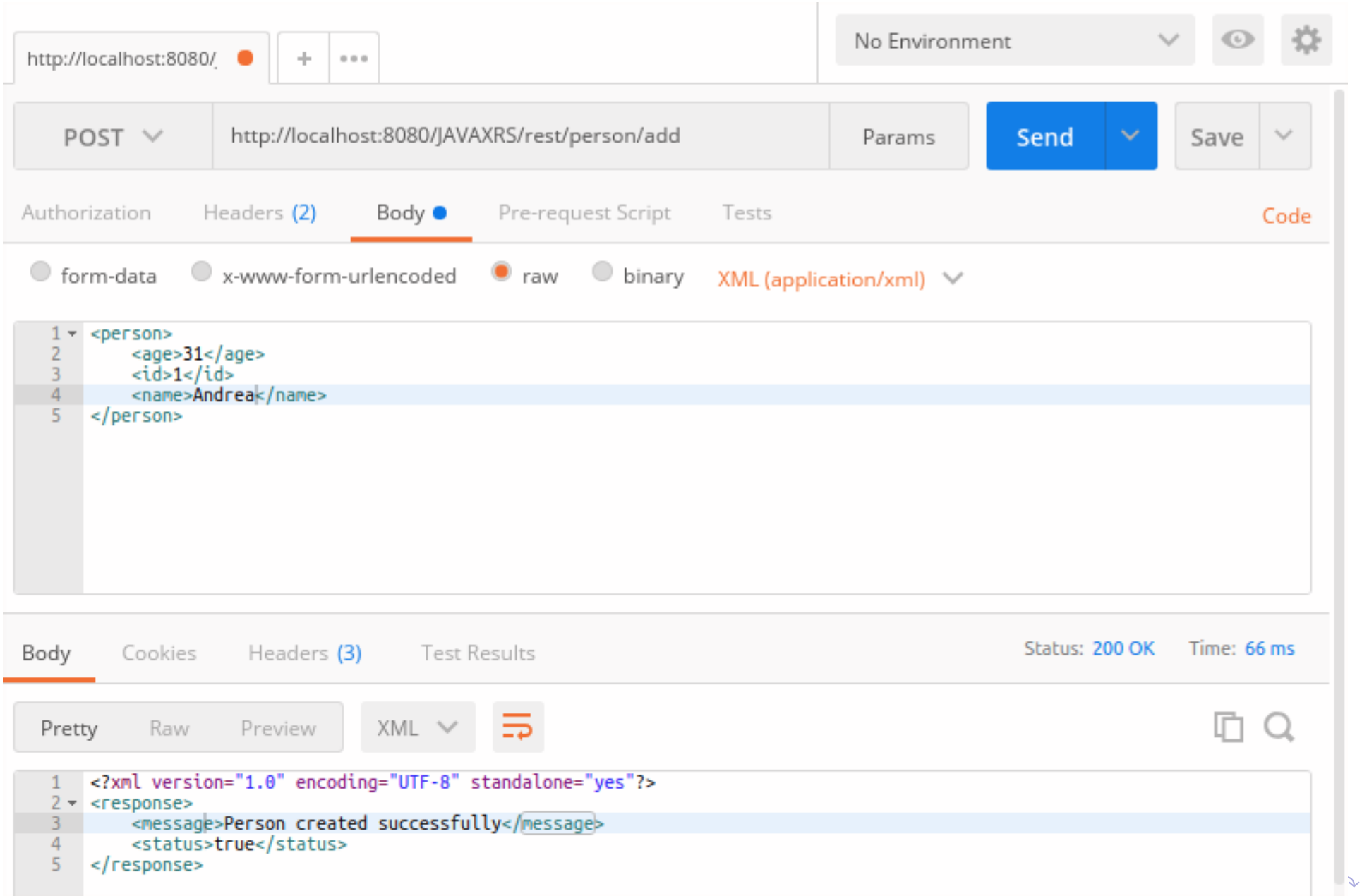
Below are some of the tests performed using Postman chrome extension for this web service. Note that we have to provide Accept and Content-Type values as application/xml in request header as shown in below image.



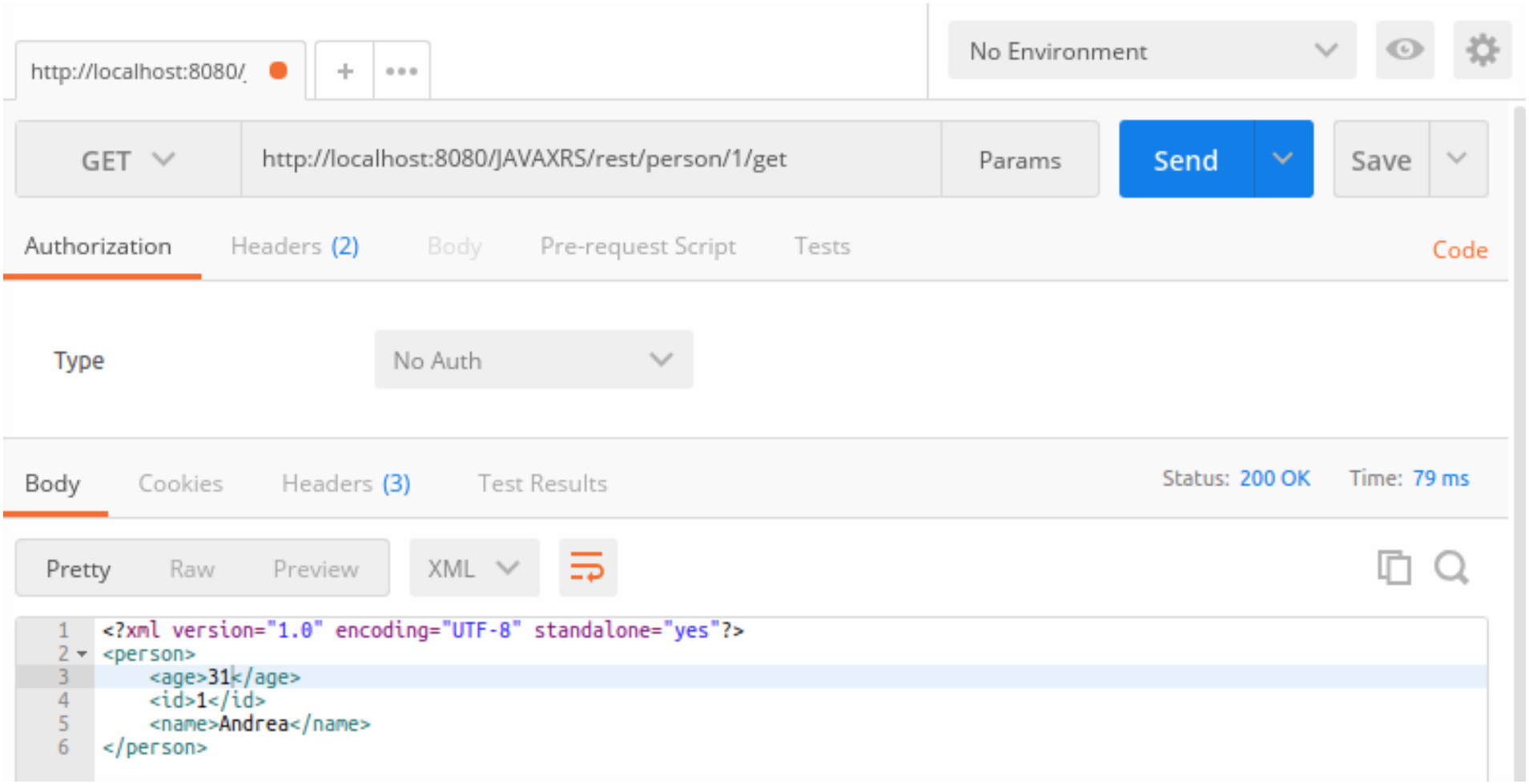
getDummy


http://localhost:8080/
No Environment
GET
http://localhost:8080/JAVAXRS/rest/person/0/getDummy
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Key
Value
Description
Bulk Edit
Presets
Accept
application/xml
Content-Type
application/xml
New key
Value
Description
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 35 ms
Pretty
Raw
Preview
XML
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person>
<age>99</age>
<id>0</id>
<name>Dummy</name>
</person>

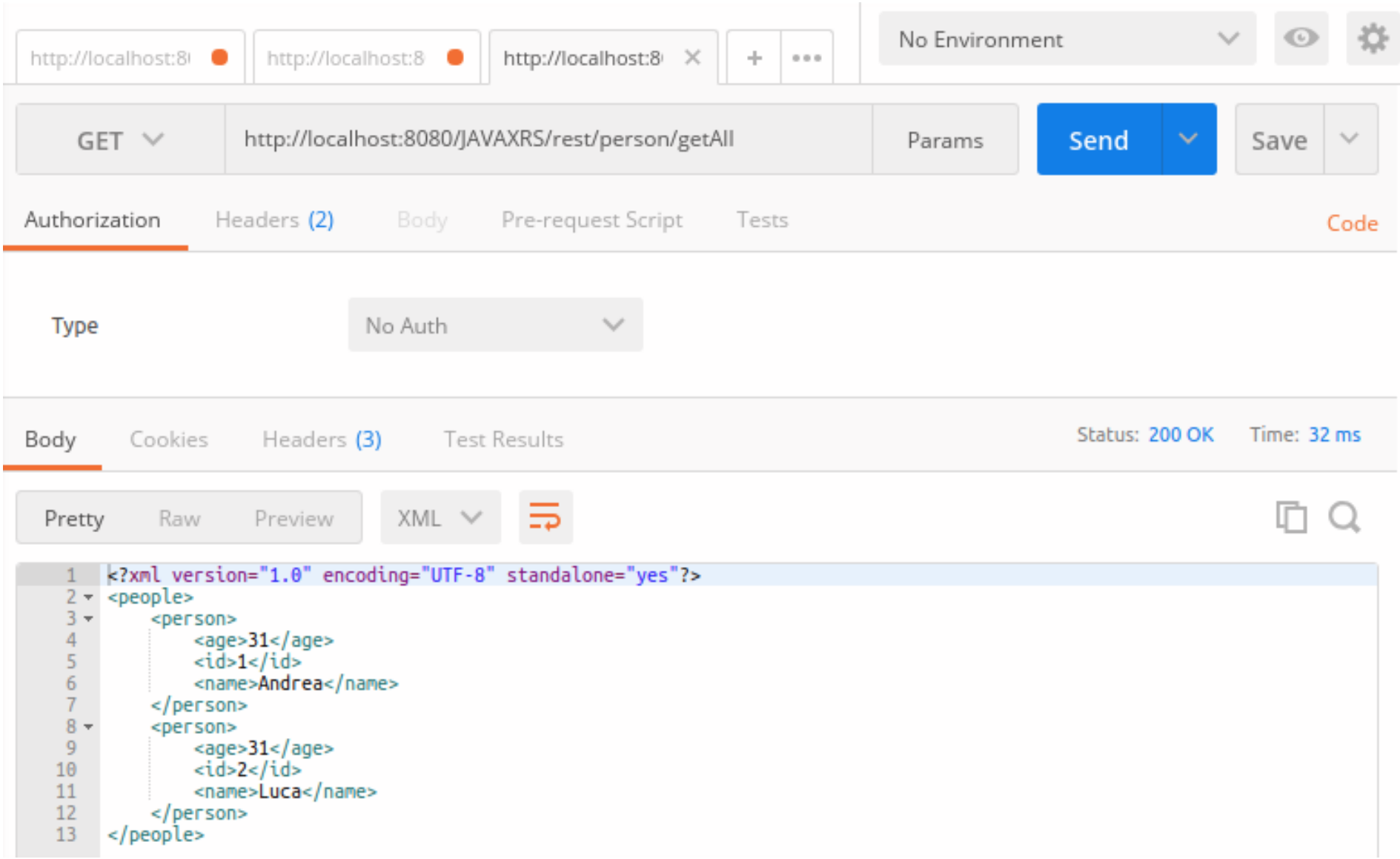
Add


http://localhost:8080/
No Environment
POST
http://localhost:8080/JAVAXRS/rest/person/add
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
XML (application/xml)
1 <person>
2 <age>31</age>
3 <id>1</id>
4 <name>Andrea</name>
5 </person>
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 66 ms
Pretty
Raw
Preview
XML
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <response>
3 <message>Person created successfully</message>
4 <status>true</status>
5 </response>

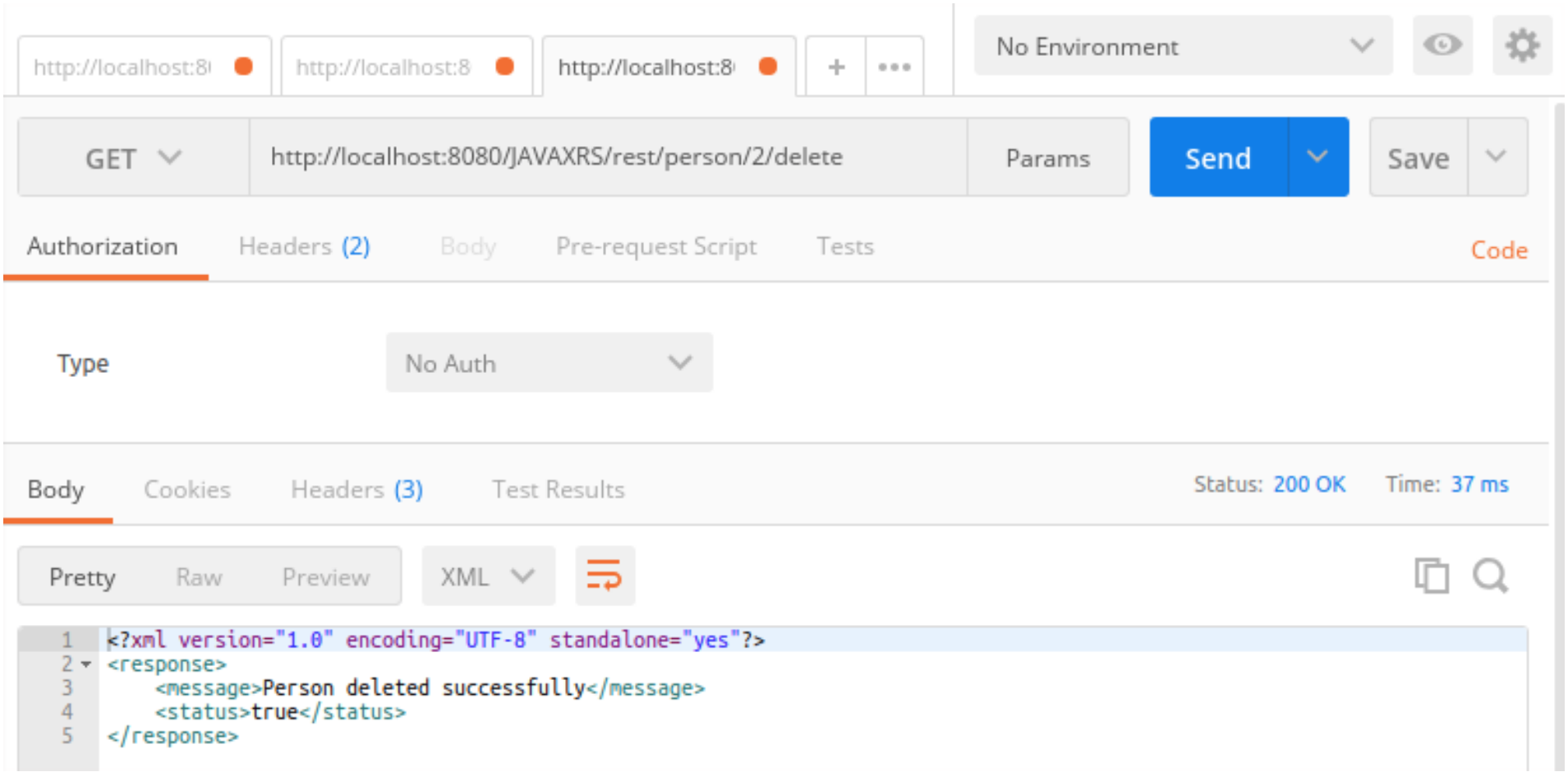
Get


http://localhost:8080/
No Environment
GET
http://localhost:8080/JAVAXRS/rest/person/1/get
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 79 ms
Pretty
Raw
Preview
XML
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person>
<age>31</age>
<id>1</id>
<name>Andrea</name>
</person>

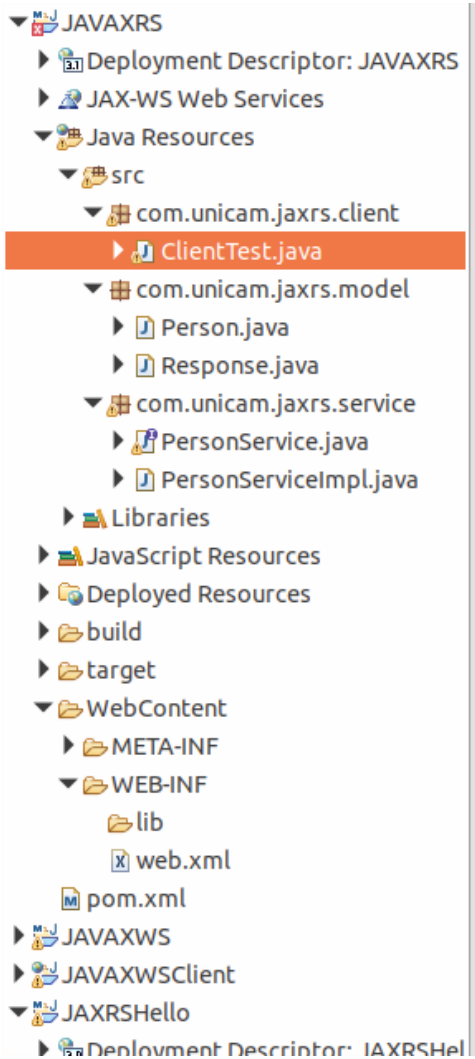
getAll


No Environment
GET
http://localhost:8080/JAVAXRS/rest/person/getAll
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 32 ms
Pretty
Raw
Preview
XML
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<people>
<person>
<age>31</age>
<id>1</id>
<name>Andrea</name>
</person>
<person>
<age>31</age>
<id>2</id>
<name>Luca</name>
</person>
</people>

Delete


No Environment
GET
http://localhost:8080/JAVAXRS/rest/person/2/delete
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 37 ms
Pretty
Raw
Preview
XML
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
<message>Person deleted successfully</message>
<status>true</status>
</response>

Java Client




```
1  package com.unicam.jaxrs.client;
2
3  import java.net.URI;
13
14 public class ClientTest {
15
16   public static void main(String[] args) {
17
18             Client client = ClientBuilder.newClient();
19             WebTarget target = client.
20                       target("http://localhost:8080/JAVAXRS/rest/person");
21
22
23
24
25 Person testPerson=new Person();
26 testPerson.setId(0);
27 testPerson.setAge(31);
28 testPerson.setName("Andrea");
29
30 //Used to add a person
31 String response = target.path("add").request()
32 .post(Entity.entity(testPerson.getXML(), MediaType.APPLICATION_XML)
33           , String.class);
34 System.out.println("customer created with id: "+response);
35
36 //Uset to get all persons
37 System.out.println(target.path("getAll").request().get(String.class));
38
39 //Used to get a person
40 System.out.println(target.path("0").path("get").request().get(String.class));
41
42   }
```

For Further Research Material

https://www.logicbig.com/tutorials/java-ee-tutorial/jax-rs/getting-started-with-jax-rs.html

SOAP & WSDL

SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, **it's platform and language independent**. So our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.

WSDL stands for Web Service Description Language. WSDL is an XML based document that provides technical details about the web service. Some of the useful information in WSDL document are: **method name, port types, service end point, binding, method parameters** etc.

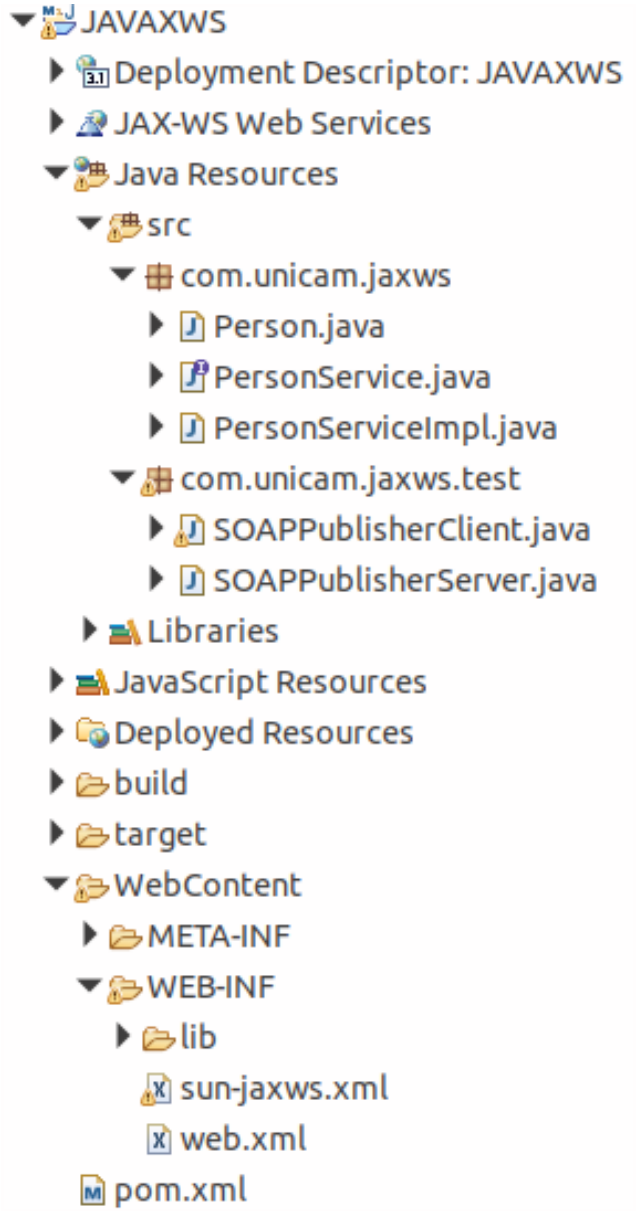
Addvantages of Web Service

- **Interoperability**: Because web services work over network and use XML technology to communicate, it can be developed in any programming language supporting web services development.
- **Reusability**: One web service can be used by many client applications at the same time. For example, we can expose a web service for technical analysis of a stock and it can be used by all the banks and financial institutions.
- **Loose Coupling**: Web services client code is totally independent with server code, so we have achieved loose coupling in our application. This leads to easy maintenance and easy to extend.
- **Easy** to deploy and integrate
- **Multiple** service versions can be running at same time.

JAX-WS

JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to **build web services server and client application**. It's **part of standard Java API**, so we don't need to include anything else which working with it.

Project


JAVAXWS
Deployment Descriptor: JAVAXWS
JAX-WS Web Services
Java Resources
src
com.unicam.jaxws
Person.java
PersonService.java
PersonServiceImpl.java
com.unicam.jaxws.test
SOAPPublisherClient.java
SOAPPublisherServer.java
Libraries
JavaScript Resources
Deployed Resources
build
target
WebContent
META-INF
WEB-INF
lib
sun-jaxws.xml
web.xml
pom.xml

Person.java

```java
package com.unicam.jaxws;

import java.io.Serializable;

public class Person implements Serializable{

    private static final long serialVersionUID = -5577579081118070434L;

    private String name;
    private int age;
    private int id;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Override
    public String toString(){
        return id+"::"+name+"::"+age;
    }

}
```

PersonService Interface

```java
package com.unicam.jaxws;

import javax.jws.WebMethod;


@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface PersonService {

    @WebMethod
    public boolean addPerson(Person p);

    @WebMethod
    public boolean deletePerson(int id);

    @WebMethod
    public Person getPerson(int id);

    @WebMethod
    public Person[] getAllPersons();
}
```

Notice the use of **@WebService** and **@SOAPBinding** annotations from JAX-WS API. We can create SOAP web services in **RPC style** or **Document style**. We can use any of these styles to create web services, the different is seen in the way WSDL file is generated.

PersonService Implementation

Most important part is the **@WebService** annotation where we are providing **endpointInterface** value as the interface we have for our web service. This way JAX-WS know the class to use for implementation when web service methods are invoked.

```java
package com.unicam.jaxws;

import java.util.HashMap;

@WebService(endpointInterface = "com.unicam.jaxws.PersonService")
public class PersonServiceImpl implements PersonService {

    private static Map<Integer,Person> persons = new HashMap<Integer,Person>();

    @Override
    public boolean addPerson(Person p) {
        if(persons.get(p.getId()) != null) return false;
        persons.put(p.getId(), p);
        return true;
    }

    @Override
    public boolean deletePerson(int id) {
        if(persons.get(id) == null) return false;
        persons.remove(id);
        return true;
    }

    @Override
    public Person getPerson(int id) {
        return persons.get(id);
    }

    @Override
    public Person[] getAllPersons() {
        Set<Integer> ids = persons.keySet();
        Person[] p = new Person[ids.size()];
        int i=0;
        for(Integer id : ids){
            p[i] = persons.get(id);
            i++;
        }
        return p;
    }
}
```

Endpoint Publication

```java
package com.unicam.jaxws.test;

import javax.xml.ws.Endpoint;

public class SOAPPublisherServer {

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8888/JAVAXWS/personWS", new PersonServiceImpl());
    }
}
```

Test the Endpoint

Running the above program as java application and your web service will be published at the given endpoint in the program. We can access it's WSDL document by adding ?wsdl to the endpoint url as shown in below image.

`http://localhost:8888/JAVAXWS/personWS`

Client

```java
package com.unicam.jaxws.test;

import java.net.MalformedURLException;

public class SOAPPublisherClient {

    public static void main(String[] args) throws MalformedURLException {
        URL wsdlURL = new URL("http://localhost:8080/JAVAXWS/personWS?wsdl");
        //check above URL in browser, you should see WSDL file

        //creating QName using targetNamespace and name
        QName qname = new QName("http://jaxws.unicam.com/", "PersonServiceImplService");

        Service service = Service.create(wsdlURL, qname);

        //We need to pass interface and model beans to client
        PersonService ps = service.getPort(PersonService.class);

        Person p1 = new Person(); p1.setName("Pankaj"); p1.setId(1); p1.setAge(30);
        Person p2 = new Person(); p2.setName("Meghna"); p2.setId(2); p2.setAge(25);

        //add person
            System.out.println("Add Person Status="+ps.addPerson(p1));

        System.out.println("Add Person Status="+ps.addPerson(p2));

        //get person
        System.out.println(ps.getPerson(1));

        //get all persons
        System.out.println(Arrays.asList(ps.getAllPersons()));

        //delete person
        System.out.println("Delete Person Status="+ps.deletePerson(2));

        //get all persons
        System.out.println(Arrays.asList(ps.getAllPersons()));
    }
}
```

Results:

```
Add Person Status=true
Add Person Status=true
1::Pankaj::30
[1::Pankaj::30, 2::Meghna::25]
Delete Person Status=true
[1::Pankaj::30]
```

```
Add Person Status=false
Add Person Status=true
1::Pankaj::30
[1::Pankaj::30, 2::Meghna::25]
Delete Person Status=true
[1::Pankaj::30]
```

JAX-WS Client Program

Web services just expose WSDL and third party applications don't have access to these classes. So in that case, we can use **wsimport** utility to generate the client stubs. This utility comes with standard installation of JDK. wsimport -s . http://localhost:8888/JAVAXWS/personWS?wsdl

Client Class

```
import java.util.Arrays;

import com.unicam.jaxws.Person;
import com.unicam.jaxws.PersonService;
import com.unicam.jaxws.PersonServiceImplService;

public class TestPersonService {

    public static void main(String[] args) {

        PersonServiceImplService serviceImpl = new PersonServiceImplService();

        PersonService service = serviceImpl.getPersonServiceImplPort();

        Person p1 = new Person(); p1.setName("Pankaj"); p1.setId(1); p1.setAge(30);
        Person p2 = new Person(); p2.setName("Meghna"); p2.setId(2); p2.setAge(25);

        System.out.println("Add Person Status="+service.addPerson(p1));
        System.out.println("Add Person Status="+service.addPerson(p2));

      //get person
        System.out.println(service.getPerson(1));

        //get all persons
        System.out.println(Arrays.asList(service.getAllPersons()));

        //delete person
        System.out.println("Delete Person Status="+service.deletePerson(2));

        //get all persons
        System.out.println(Arrays.asList(service.getAllPersons()));

    }
```

Dependecies

Considering the previous example transform the project into a maven project and add the following dependecies.

```
<dependencies>
    <dependency>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-rt</artifactId>
        <version>2.2.10</version>
    </dependency>
</dependencies>
```

web.xml

Next steps are required to create the project as a web archive that we can deploy in servlet container. Add the following web.xml file.

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    id="WebApp_ID" version="3.1">
    <display-name>JAXWS-Tomcat</display-name>

    <listener>
        <listener-class>
            com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>JAXWSServlet</servlet-name>
        <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>JAXWSServlet</servlet-name>
        <url-pattern>/personWS</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

sun-jaxws.xml

Next step is to create sun-jaxws.xml file inside WEB-INF directory where we will provide endpoint details. url-pattern should be same as defined in the web.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
  <endpoint
      name="PersonServiceImpl"
      implementation="com.unicam.jaxws.PersonServiceImpl"
      url-pattern="/personWS"/>
</endpoints>
```

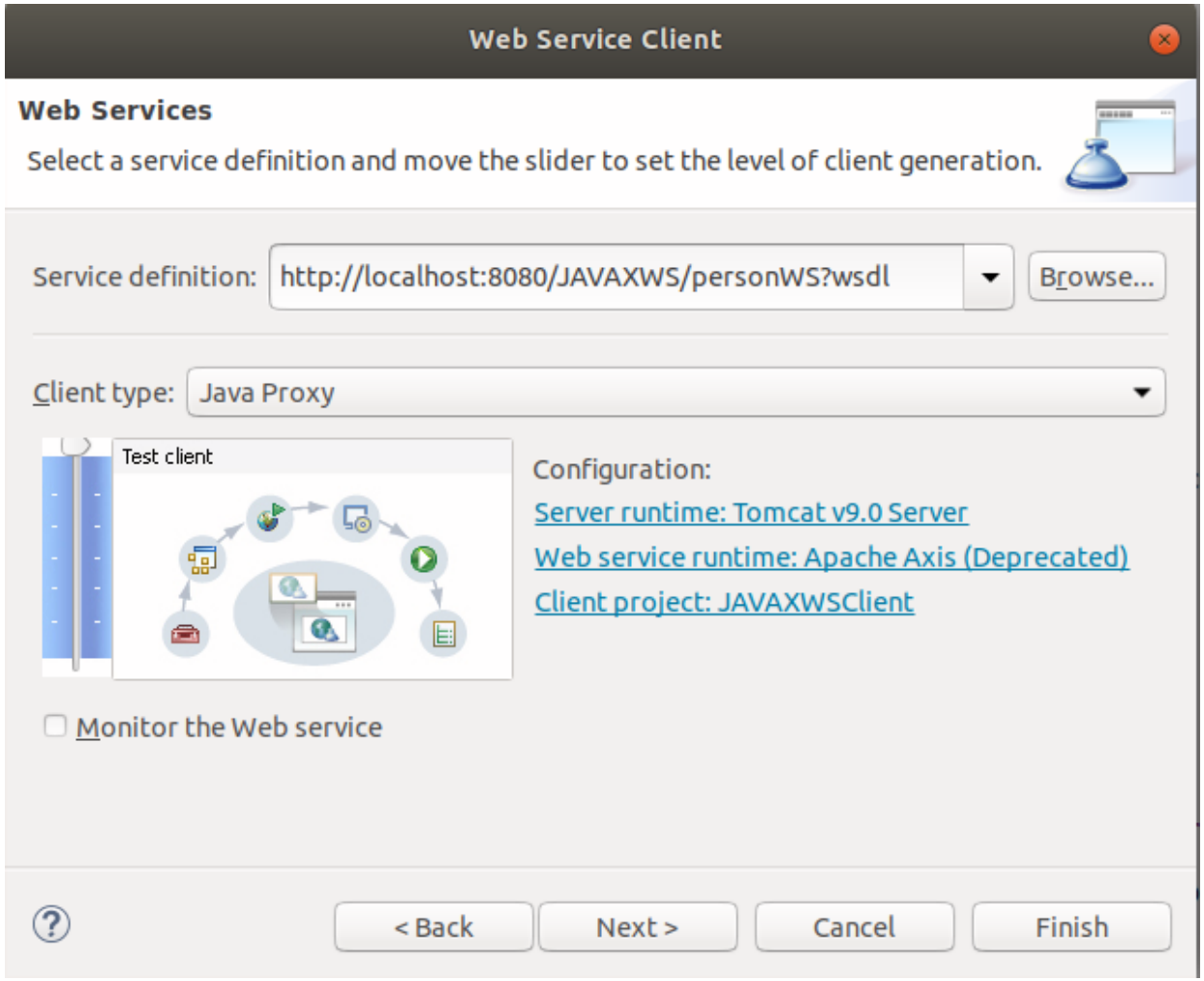
Web service run

Execute the project as a server application and chack the corresponding link:

`http://localhost:8080/JAVAXWS/personWS`

Automatic Client Generation using Eclipse

Create a new Web Service Client



Test web Service from web

Run the project as a server application and test it using web application.

`http://localhost:8080/JAVAXWSClient/samplePersonServiceProxy/TestClient.jsp`