Web Service

By

Praveen Oruganti
Linktree: https://linktr.ee/praveenoruganti
Email: praveenorugantitech@gmail.com

Web Service

A Web Service can be defined by following ways

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

For example, java application can interact with Java, .Net and PHP applications.
So web service is a language independent way of communication.

Before going to in-depth of web services, let's see basics of XML and XSD

XML (Extensible Markup Language):

- ✓ Generally XML is used for 2 things i.e... 1. Configuration File (web.xml, server.xml, pom.xml, build.xml, beans.xml etc) 2. Data Exchange 3. To save and manipulate and present the data to the client.
- ✓ XML carries both the data and metadata.
- ✓ XML follows Well-Formedness and has validation rules as well.

XSD (XML Schema Definition)

- ✓ XSD defines the **Grammer** or **Blueprint** of an xml document i.e... we can use it to mention what **Elements, Attributes** can be there in xml and what **Namespaces** can be used in xml, **order** it maintains, **number of occurrences** and **Restrictions.**
- ✓ XML follows XSD then it's a Valid XML document.
- ✓ XSD is also an xml file rather its extension is .xsd and all elements which are used in XSD are provided by W3C (World Wide Web Consortium)
- ✓ An XSD is a formal contract that specifies how an XML document can be formed. It is often used to validate an XML document, or to generate code from.

Namespaces

Namespaces allows us to uniquely identify the elements and attributes of an XML document.
When we are creating a schema, first we will be declaring the targetNamespace: https://praveenorugantitech.blogspot.com which will be unique and in order to use the namespace for element we can use the prefix like
xmlns:op="https://praveenorugantitech.blogspot.com"

For example,
<blogspot xmlns:op="https://praveenoruganti tech.blogspot.com">
<op:authorName>Praveen Oruganti</op.authorName>
<op:publisheddate>19-10-2019</op:publisheddate>
</blogspot>

XML Schema- Use Case

Patient Data and Clinical Data

Here for Patient.xsd, we have created complexType Patient.

Patient.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://praveenoruganti.blogspot.com/Patient"
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    elementFormDefault="qualified">
    <element name="Patient" type="tns:Patient" />
    <complexType name="Patient">
        <sequence>
            <element name="id" type="int" />
            <element name="Name" type="string" />
            <element name="Age" type="int" />
            <element name="DOB" type="date" />
            <element name="Email" type="string" />
            <element name="Gender" type="string" />
            <element name="Phone" type="string" />
        </sequence>
    </complexType>
</schema>
```

Patient.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:Patient
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://praveenoruganti.blogspot.com/Patient Patient.xsd ">
    <tns:id>1</tns:id>
    <tns:Name>Praveen Oruganti</tns:Name>
    <tns:Age>34</tns:Age>
    <tns:DOB>2001-01-01</tns:DOB>
    <tns:Email>praveenoruganti@gmail.com</tns:Email>
    <tns:Gender>M</tns:Gender>
    <tns:Phone>9999999999</tns:Phone>
</tns:Patient>
```

Now let's see how we can create simple type using restrictions.

Patient.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://praveenoruganti.blogspot.com/Patient"
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    elementFormDefault="qualified">
    <element name="Patient" type="tns:Patient" />
    <complexType name="Patient">
        <sequence>
            <element name="id" type="tns:ID" />
            <element name="Name" type="tns:String15Chars" />
            <element name="Age" type="int" />
            <element name="DOB" type="date" />
            <element name="Email" type="string" />
            <element name="Gender" type="tns:Gender" />
            <element name="Phone" type="string" />
        </sequence>
    </complexType>
    <simpleType name="ID">
        <restriction base="int">
            <pattern value="[0-9]" />
        </restriction>
    </simpleType>
    <simpleType name="String15Chars">
        <restriction base="string">
            <maxLength value="15" />
        </restriction>
    </simpleType>
    <simpleType name="Gender">
        <restriction base="string">
            <enumeration value="M" />
            <enumeration value="F" />
        </restriction>
    </simpleType>
</schema>
```

Patient.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:Patient xmlns:tns="https://praveenoruganti.blogspot.com/Patient" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="https://praveenorug
    <tns:id>1</tns:id>
    <tns:Name>PraveenOruganti</tns:Name>
    <tns:Age>34</tns:Age>
    <tns:DOB>2001-01-01</tns:DOB>
    <tns:Email>praveenoruganti@gmail.com</tns:Email>
    <tns:Gender>M</tns:Gender>
    <tns:Phone>1234567890</tns:Phone>
</tns:Patient>
```

Now let's see how to control order and number of occurrences by creating a new complex type PaymentType. Previously we have seen sequence but now we will see choice and all.

To control the number of occurrences we need to use min occurs and maxoccurs.

Patient.xsd

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://praveenoruganti.blogspot.com/Patient"
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    elementFormDefault="qualified">
    <element name="Patient" type="tns:Patient" />
    <complexType name="Patient">
        <sequence>
            <element name="id" type="tns:ID" minOccurs="0" />
            <element name="Name" type="tns:String15Chars" />
            <element name="Age" type="int" />
            <element name="DOB" type="date" />
            <element name="Email" type="string" maxOccurs="unbounded" />
            <element name="Gender" type="tns:Gender" />
            <element name="Phone" type="string" maxOccurs="2" />
            <element name="payment" type="tns:PaymentType" />
        </sequence>
    </complexType>
    <complexType name="PaymentType">
        <choice>
            <element name="cash" type="int"></element>
            <element name="insurance" type="tns:Insurance"></element>
        </choice>
    </complexType>
    <complexType name="Insurance">
        <all>
            <element name="provider" type="string" />
            <element name="limit" type="int" />
        </all>
    </complexType>
    <simpleType name="ID">
        <restriction base="int">
            <pattern value="[0-9]" />
        </restriction>
    </simpleType>
</simpleType>
    <simpleType name="String15Chars">
        <restriction base="string">
            <maxLength value="15" />
        </restriction>
    </simpleType>
    <simpleType name="Gender">
        <restriction base="string">
            <enumeration value="M" />
            <enumeration value="F" />
        </restriction>
    </simpleType>
</schema>
```

Patient.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:Patient
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://praveenoruganti.blogspot.com/Patient Patient.xsd ">
    <tns:Name>PraveenOruganti</tns:Name>
    <tns:Age>34</tns:Age>
    <tns:DOB>2001-01-01</tns:DOB>
    <tns:Email>praveenoruganti@gmail.com</tns:Email>
    <tns:Email>orugantipraveen@gmail.com</tns:Email>
    <tns:Gender>M</tns:Gender>
    <tns:Phone>1234567890</tns:Phone>
    <tns:Phone>1234567891</tns:Phone>
    <tns:payment>
        <tns:insurance>
            <tns:provider>HDFC</tns:provider>
            <tns:limit>100000</tns:limit>
        </tns:insurance>
    </tns:payment>
</tns:Patient>
```

elementFormDefault

elementFormDefault="qualified" means it's always qualified with namespace for example if you see the xml, tns namespace is present for all the elements.
If elementFormDefault="unqualified" means there is no need to give namespace for the elements.
It is good practice to have elementFormDefault="qualified".

Let's see how to create attribute.

Patient.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://praveenoruganti.blogspot.com/Patient"
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    elementFormDefault="qualified">
    <element name="Patient" type="tns:Patient" />
    <complexType name="Patient">
        <sequence>
            <element name="Name" type="tns:String15Chars" />
            <element name="Age" type="int" />
            <element name="DOB" type="date" />
            <element name="Email" type="string" maxOccurs="unbounded" />
            <element name="Gender" type="tns:Gender" />
            <element name="Phone" type="string" maxOccurs="2" />
            <element name="payment" type="tns:PaymentType" />
        </sequence>
        <attribute name="id" type="tns:ID"></attribute>
    </complexType>
```

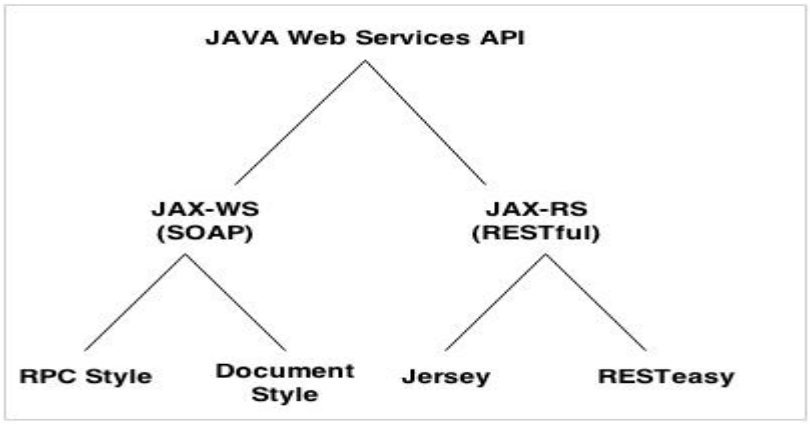
Patient.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:Patient
    xmlns:tns="https://praveenoruganti.blogspot.com/Patient"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://praveenoruganti.blogspot.com/Patient Patient.xsd "
    id="1">
    <tns:Name>PraveenOruganti</tns:Name>
    <tns:Age>34</tns:Age>
    <tns:DOB>2001-01-01</tns:DOB>
    <tns:Email>praveenoruganti@gmail.com</tns:Email>
    <tns:Email>orugantipraveen@gmail.com</tns:Email>
    <tns:Gender>M</tns:Gender>
    <tns:Phone>1234567890</tns:Phone>
    <tns:Phone>1234567891</tns:Phone>
    <tns:payment>
        <tns:insurance>
            <tns:provider>HDFC</tns:provider>
            <tns:limit>100000</tns:limit>
        </tns:insurance>
    </tns:payment>
</tns:Patient>
```

You can view this code in my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-xsd-master)

Types of Web Services

There are mainly two types of web services.
1. SOAP web services. -> JAX-WS implementation. There are two ways to write JAX-WS application code: by RPC style and Document style.
2. RESTful web services. -> JAX-RS implementation. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.



SOAP Webservice

Pros

- ✓ They are platform independent
  - ➢ HTTP- Transport Independent
  - ➢ XML- Data Independent

- ✓ Application Tailoring/Customization
- ✓ Legacy Application are Great! (HTTP and XML)
- ✓ New Revenue/Profit Channels
- ✓ Firewalls like Web services

Cons

- ✓ Ambiguous Web services Standards.
- ✓ Performance impact due to serialization and deserialization of SOAP messages

SOAP web services are used when

- ✓ Formal contract is required -> WSDL
- ✓ Non Functional Requirements like Security and Transaction Management. It provides JAXWS standards. For example Apache CXF provides standards for this and we can easily configure this as it comes out of box.
- ✓ Reliable Asynchronous Processing

Web Service Components

Below are the major web service components.

- ✓ SOAP--Simple Object Access Protocol
- ✓ WSDL-Web Service Description language
- ✓ UDDI--Universal Description, Discovery, and Integration
- ✓ XML- Extensible Markup Language

SOAP

- ✓ SOAP is an acronym for Simple Object Access Protocol.
- ✓ SOAP is a XML-based protocol for accessing web services.
- ✓ SOAP is a W3C recommendation for communication between applications.
- ✓ SOAP is XML based, so it is platform independent and language independent. In other words, it can be used with Java, .Net or PHP language on any platform.

SOAP Message Structure

One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.

Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

The SOAP message is nothing but a mere XML document which has the below components.

- ✓ An Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
- ✓ A Header element that contains header information – The header element can contain information such as authentication credentials which can be used by the calling application. It can also contain the definition of complex types which could be used in the SOAP message. By default, the SOAP message can contain parameters which could be of simple types such as strings and numbers, but can also be a complex object type.
- ✓ Body element that contains call and response information – This element is what contains the actual data which needs to be sent between the web service and the calling application.

A simple SOAP Message has the following elements –

- ✓ The Envelope element
- ✓ The header element and
- ✓ The body element
- ✓ The Fault element (Optional)

SOAP Envelope Element

The first bit of the building block is the SOAP Envelope.

The SOAP Envelope is used to encapsulate all of the necessary details of the SOAP messages, which are exchanged between the web service and the client application.

The SOAP envelope element is used to indicate the beginning and end of a SOAP message. This enables the client application which calls the web service to know when the SOAP message ends.

The following points can be noted on the SOAP envelope element.

- ✓ Every SOAP message needs to have a root Envelope element. It is absolutely mandatory for SOAP message to have an envelope element.
- ✓ Every Envelope element needs to have at least one soap body element.

- ✓ If an Envelope element contains a header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the body element.
- ✓ The envelope changes when SOAP versions change.
- ✓ A v1.1-compliant SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace.
- ✓ A v1.2-compliant SOAP processor generates a Version Mismatch fault if it receives a message that does not include the v1.2 envelope namespace.

The Fault message

When a request is made to a SOAP web service, the response returned can be of either 2 forms which are a successful response or an error response. When a success is generated, the response from the server will always be a SOAP message. But if SOAP faults are generated, they are returned as "HTTP 500" errors.

The SOAP Fault message consists of the following elements.

1. **<faultCode>**- This is the code that designates the code of the error. The fault code can be either of any below values
   - ✓ SOAP-ENV:VersionMismatch – This is when an invalid namespace for the SOAP Envelope element is encountered.
   - ✓ SOAP-ENV:MustUnderstand - An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood.
   - ✓ SOAP-ENV:Client - The message was incorrectly formed or contained incorrect information.
   - ✓ SOAP-ENV:Server - There was a problem with the server, so the message could not proceed.
2. **<faultString>** - This is the text message which gives a detailed description of the error.
3. **<faultActor> (Optional)**- This is a text string which indicates who caused the fault.
4. **<detail>(Optional)** - This is the element for application-specific error messages. So the application could have a specific error message for different business logic scenarios.

For example,

soap.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope>
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
        <!-- Generally Security Information is passed i.e.. meta information -->
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>praveen</wsse:Username>
                <wsse:Password>java</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <soap:CreditCardReqeuest>

        </soap:CreditCardReqeuest>
        <soap:Fault>
            <!-- Fault Information -->
            <soap:code>soap:Server</soap:code>
            <soap:Reason>
                <soap:text>
                    Card Expired
                </soap:text>

            </soap:Reason>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

SOAP Communication Model

All communication by SOAP is done via the HTTP protocol. Prior to SOAP, a lot of web services used the standard RPC (Remote Procedure Call) style for communication. This was the simplest type of communication, but it had a lot of limitations.

Let's consider the below diagram to see how this communication works. In this example, let's assume the server hosts a web service which provided 2 methods as

- ✓ **GetEmployee** - This would get all Employee details
- ✓ **SetEmployee** – This would set the value of the details like employees dept, salary, etc. accordingly.

In the normal RPC style communication, the client would just call the methods in its request and send the required parameters to the server, and the server would then send the desired response.

The above communication model has the below serious limitations

1. **Not Language Independent** – The server hosting the methods would be in a particular programming language and normally the calls to the server would be in that programming language only.
2. **Not the standard protocol** – When a call is made to the remote procedure, the call is not carried out via the standard protocol. This was an issue since mostly all communication over the web had to be done via the HTTP protocol.
3. **Firewalls** – Since RPC calls do not go via the normal protocol, separate ports need to be open on the server to allow the client to communicate with the server. Normally all firewalls would block this sort of traffic, and a lot of configuration was generally required to ensure that this sort of communication between the client and the server would work.

To overcome all of the limitations cited above, SOAP would then use the below communication model

1. The client would format the information regarding the procedure call and any arguments into a SOAP message and sends it to the server as part of an HTTP request. This process of encapsulating the data into a SOAP message was known as **Marshalling.**
2. The server would then unwrap the message sent by the client, see what the client requested for and then send the appropriate response back to the client as a SOAP message. The practice of unwrapping a request sent by the client is known as **Demarshalling.**

Summary

- SOAP is a protocol which is used to interchange data between applications which are built on different programming languages.
- SOAP is built upon the XML specification and works with the HTTP protocol. This makes it a perfect for usage within web applications.
- The SOAP building blocks consist of a SOAP Message. Each SOAP message consists of an envelope element, a header, and a body element.
- The envelope element is the mandatory element in the SOAP message and is used to encapsulate all of the data in the SOAP message.
- The header element can be used to contain information such as authentication information or the definition of complex data types.
- The body element is the main element which contains the definition of the web methods along with any parameter information if required.

WSDL

WSDL is an XML-based file which basically tells the client application what the web service does. It is known as the Web Services Description Language (WSDL).

- ✓ WSDL is an acronym for Web Services Description Language.
- ✓ WSDL is an xml document containing information about web services such as method name, method parameter and how to access it.
- ✓ WSDL is a part of UDDI. It acts as an interface between web service applications.
- ✓ WSDL is pronounced as "wiz-dull".
- ✓ It is a contract between provider and consumer.

Structure of a WSDL Document

A WSDL document is used to describe a web service. This description is required, so that client applications are able to understand what the web service actually does.

- ✓ The WSDL file contains the location of the web service and The methods which are exposed by the web service.
- ✓ The WSDL file itself can look very complex to any user, but it contains all the necessary information that any client application would require to use the relevant web service.

WSDL document contains the following elements:

- ✓ Definition
- ✓ Data types
- ✓ Message
- ✓ Operation
- ✓ Port types
- ✓ Binding
- ✓ Port
- ✓ Service

WSDL mainly comprises of

definitions
types
messages
operation
porttype

Abstract Section:

- ✓ Wsdl:types – Has the xml schema that can be used to build the messages
- ✓ Wsdl:messages: the request and response xml messages built using the schema in the types section above.
- ✓ Wsdl:Operation: Each operation that the web service exposes.
- ✓ Wsdl:PortType: Groups the operations or web service methods together.

Physical Portion:

- ✓ Wsdl:Binding – We define the binding for all the operations which controls how the soap message is generated.
- ✓ Wsdl:service: The location/url of the service using which it can be accessed.

For example,

UserProfile.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="https://praveenoruganti.blogspot.com/userProfile"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="UserProfileService"
    targetNamespace="https://praveenoruganti.blogspot.com/userProfile"
    xmlns:upSchema="https://praveenoruganti.blogspot.com/userProfile/schema/UserProfile.xsd">
    <wsdl:types>
        <xsd:schema
            targetNamespace="https://praveenoruganti.blogspot.com/userProfile"
            elementFormDefault="qualified">
            <xsd:import
                namespace="https://praveenoruganti.blogspot.com/userProfilee/schema/UserProfile.xsd"
                schemaLocation="UserProfile.xsd" />
            <xsd:element name="GetUserProfile">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="userName" type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="GetUserProfileResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="UserProfile"
                            type="upSchema:UserProfile" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>

    <wsdl:message name="GetUserProfileRequest">
        <wsdl:part name="params" element="tns:GetUserProfile" />
    </wsdl:message>
    <wsdl:message name="GetUserProfileResponse">
        <wsdl:part name="result"
            element="tns:GetUserProfileResponse" />
    </wsdl:message>

    <wsdl:portType name="UserProfilePortType">
        <wsdl:operation name="GetUserProfile">
            <wsdl:input message="tns:GetUserProfileRequest" />
            <wsdl:output message="tns:GetUserProfileResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="UserProfileBinding"
        type="tns:UserProfilePortType">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="GetUserProfile">
            <soap:operation soapAction="urn:GetUserProfile" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="UserProfileService">
        <wsdl:port binding="tns:UserProfileBinding"
            name="UserProfilePort">
            <soap:address
                location="http://localhost/services/UserProfileService" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

WSDL Styles

A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives you four style/use models:

- ✓ RPC/encoded
- ✓ RPC/literal
- ✓ Document/encoded
- ✓ Document/literal

For example, I have a Java method
public void myMethod(int x, float y);

RPC/encoded

Using Java-to-WSDL tool, let's see how wsdl will be generated for RPC/encoded

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- I won't bother with the details, just assume it's RPC/encoded. -->
```

At runtime webservice generated i.e.. Apache Axis or CXF generates the below soap message

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Strengths

- ✓ The WSDL is about as straightforward as it's possible for WSDL to be.

- ✓ The operation name appears in the message, so the receiver has an easy time dispatching this message to the implementation of the operation.

Weaknesses

- ✓ The type encoding info (such as xsi:type="xsd:int") is usually just overhead which degrades throughput performance.
- ✓ You cannot easily validate this message since only the <x ...>5</x> and <y ...>5.0</y> lines contain things defined in a schema; the rest of the soap:body contents comes from WSDL definitions.
- ✓ Although it is legal WSDL, RPC/encoded is not WS-I compliant.

RPC/literal

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- I won't bother with the details, just assume it's RPC/<strong>literal</strong>.

<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Strengths

- ✓ The WSDL is still about as straightforward as it is possible for WSDL to be.
- ✓ The operation name still appears in the message.
- ✓ The type encoding info is eliminated.
- ✓ RPC/literal is WS-I compliant.

Weaknesses

You still cannot easily validate this message since only the <x ...>5</x> and <y ...>5.0</y> lines contain things defined in a schema; the rest of the soap:body contents comes from WSDL definitions.

Document/encoded

Nobody follows this style. It is not WS-I compliant. So let's move on.

Document/literal

```
<strong><types>
    <schema>
        <element name="xElement" type="xsd:int"/>
        <element name="yElement" type="xsd:float"/>
    </schema>
</types></strong>

<message name="myMethodRequest">
    <part name="x" <strong>element="xElement"</strong>/>
    <part name="y" <strong>element="yElement"</strong>/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- I won't bother with the details, just assume it's <strong>document</strong>/lite
```

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

Strengths

- ✓ There is no type encoding info.
- ✓ You can finally validate this message with any XML validator. Everything within the soap:body is defined in a schema.
- ✓ Document/literal is WS-I compliant, but with restrictions (see weaknesses).

Weaknesses

- ✓ The WSDL is getting a bit more complicated. This is a very minor weakness, however, since WSDL is not meant to be read by humans.
- ✓ The operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible.
- ✓ WS-I only allows one child of the soap:body in a SOAP message. As you can see in Listing 7, this example's soap:body has two children.

The document/literal style seems to have merely rearranged the strengths and weaknesses from the RPC/literal model. You can validate the message, but you lose the operation name. Is there anything you can do to improve upon this? Yes. It's called the document/literal wrapped pattern.

Document/literal wrapped

```
<types>
    <schema>
        <strong><element name="myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int"/>
                    <element name="y" type="xsd:float"/>
                </sequence>
            </complexType>
        </element>
        <element name="myMethodResponse">
            <complexType/>
        </element></strong>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="<strong>parameters</strong>" element="<strong>myMethod</strong>"/>
</message>
<strong><message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message></strong>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- I won't bother with the details, just assume it's document/literal. -->
```

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

These are the basic characteristics of the document/literal wrapped pattern:

- ✓ The input message has a single part.
- ✓ The part is an element.
- ✓ The element has the same name as the operation.
- ✓ The element's complex type has no attributes.

Strengths

- ✓ There is no type encoding info.
- ✓ Everything that appears in the soap:body is defined by the schema, so you can easily validate this message.
- ✓ Once again, you have the method name in the SOAP message.
- ✓ Document/literal is WS-I compliant, and the wrapped pattern meets the WS-I restriction that the SOAP message's soap:body has only one child.

Weaknesses

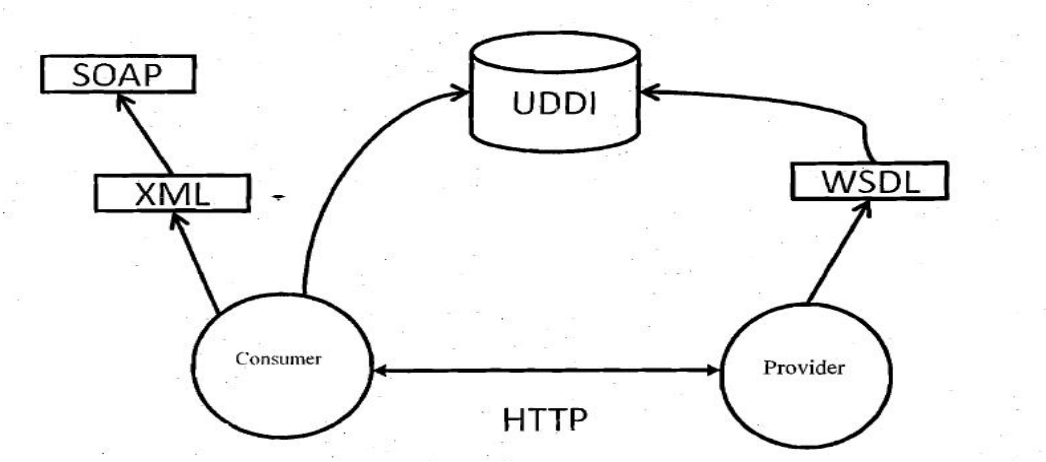
- ✓ The WSDL is even more complicated.

Summary

- ✓ A WSDL document is a document that is used to describe a web service. This is key for any client application to know where the web service is located. It also allows the client application to understand the methods available in the web service.
- ✓ The WSDL file makes it very easy for the web service to be implemented in one programming language and called from a different programming language.
- ✓ The WSDL document normally consists of a message. For each web method, there are 2 messages, one is for the input, and the other is for the output. Together they form an operation.
- ✓ WSDL files normally get created in the editor which is used for the corresponding programming language.

UDDI

- ✓ UDDI is an acronym for Universal Description, Discovery and Integration.
- ✓ UDDI is a XML based framework for describing, discovering and integrating web services.
- ✓ UDDI is a directory of web service interfaces described by WSDL, containing information about web services.



- ✓ But what services are providing by the Provider should be explained. To know the services of "Provider", "Consumer" cannot look into the code of "Provider".
- ✓ So "Consumer" needs the information like what are the services are providing, what they will take as input, and what the output they will return, what url has to call to get services ... etc.
- ✓ "Provider1' will explain all these information in one document called "WSDL".
- ✓ "WSDL" stands for Web Service Description Language
- ✓ "WSDL" is an XML document, because it should be language independent, so that any type of client can understand the services of the "Provider".
- ✓ "WSDL" is the document which explains the services information. But how can "Consumer" get that WSDL document, from where "Consumer" get?

- ✓ So "WSDL" documents has to be placed in some location from where "Consumer" can access them, that location is nothing but "UDDI"
- ✓ UDDI stands for Universal Description Discovery and Integration
- ✓ UDDI is the Registry where all the WSDL documents are registered
- ✓ UDDI should be interoperable, so it is also developed in XPIL \ I
- ✓ UDDI registry also called XML registry

SOAP web services can be implemented in 2 design approaches

- ✓ Top Down or WSDL First or Contract First
- ✓ Code First or Bottom Up

Contract First Design

- ✓ Create the WSDL file
- ✓ Generate the java stubs using tools like wsdl2java
- ✓ Implement the web services endpoint.
- ✓ **Advantages**:
  - ➢ Contract with the consumer signed off
  - ➢ Better Interoperability
  - ➢ Faster Integration

Code First Design

- ✓ Write Java Code and annotate
- ✓ Generate the WSDL from the code using java2wsdl
- ✓ **Advantages:**
  - ➢ Legacy Application Integration

Which design to choose?

- ✓ Contract First except while exposing legacy application as web services.

JAX-WS (JAVA API FOR XML BASED WEB SERVICES)

- ✓ JAX-WS provides Specification and API.
- ✓ Specification provided by rules from Oracle and these are implemented by Apache CXF and Oracle Glass Fish.
- ✓ API have set of annotations which will be used by developers.

Core Annotations -> javax.jws

1. @javax.jws.WebService which represents the endpoint
   public class OrderService
2. @javax.jws.WebMethod which represents the methods present in endpoint

@WebResult(name=”order”) Order getOrder(@WebParam(name=”orderId”) Long orderId)

3. @javax.xml.ws.WebFault which is used for custom exceptions
   MyException extends Exception
4. @javax.jws.soap.SOAPBinding -> document\literal (by default and recommended)
   We can also use other bindings and it can represent as
   @SOAPBinding(style=Style.RPC,use=Use.LITERAL)
   public interface OrderService
5. @javax.xml.ws.RequestWrapper
6. @javax.xml.ws.ResponseWrapper

JAXB (java Architecture for XML Binding)

It provides mechanism to marshal (write) java objects into XML and unmarshal (read) XML into object. Simply, you can say it is used to convert java object into xml and vice-versa.

Three tools

- ✓ XJC converts xml schema to Java classes.
- ✓ SCHEMAGEN converts Java classes to XML Schema
- ✓ RUNTIME API converts Java Objects to XML

RUNTIME API compromises of Marshalling, Unmarshall classes and Annotations
Reference Implementations are Apache CXF and current JAXB version is 2.2.

Let’s see how we can generate the java stubs from xsd using XJC maven plugin.

Patient.xsd

Patient.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://praveenoruganti.blogspot.com/Patient"
        xmlns:tns="http://praveenoruganti.blogspot.com/Patient"
        elementFormDefault="qualified">
        <element name="patient" type="tns:Patient" />
        <complexType name="Patient">
            <sequence>
                <element name="name" type="tns:String15Chars" />
                <element name="age" type="int" />
                <element name="dob" type="date" />
                <element name="email" type="string" maxOccurs="unbounded" />
                <element name="gender" type="tns:Gender" />
                <element name="phone" type="string" />
                <element name="payment" type="tns:PaymentType" />
            </sequence>
            <attribute name="id" type="tns:ID" />
        </complexType>

        <complexType name="PaymentType">
            <choice>
                <element name="cash" type="int" />
                <element name="insurance" type="tns:Insurance" />
            </choice>
        </complexType>
        <complexType name="Insurance">
            <all>
                <element name="provider" type="string" />
                <element name="limit" type="int" />
            </all>
        </complexType>
```

```
    <simpleType name="ID">
        <restriction base="int">
            <pattern value="[0-9]*"></pattern>
        </restriction>
    </simpleType>
    <simpleType name="String15Chars">
        <restriction base="string">
            <maxLength value="15" />
        </restriction>
    </simpleType>
    <simpleType name="Gender">
        <restriction base="string">
            <enumeration value="M" />
            <enumeration value="F" />
        </restriction>
    </simpleType>
</schema>
```

global.xjb

global.xjb

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jaxb:bindings version="2.0"
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  jaxb:extensionBindingPrefixes="xjc">

 <jaxb:globalBindings>
    <xjc:simple />
    <xjc:serializable uid="-1" />
    <jaxb:javaType name="java.util.Calendar" xmlType="xs:dateTime"
      parseMethod="javax.xml.bind.DatatypeConverter.parseDateTime"
      printMethod="javax.xml.bind.DatatypeConverter.printDateTime" />
  </jaxb:globalBindings>
</jaxb:bindings>
```

We are developed the xsd and xjb file.

We need to include jaxb plugin for generating the java stubs from xsd.

praveenoruganti-jaxb-xjc-master/pom.xml

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.jvnet.jaxb2.maven2</groupId>
                <artifactId>maven-jaxb2-plugin</artifactId>
                <version>0.9.0</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>generate</goal>
                        </goals>
                    </execution>
                </executions>

                <configuration>
                    <schemaDirectory>${project.basedir}/src/main/xsd</schemaDirectory>
                    <schemaIncludes>
                        <include>Patient.xsd</include>
                    </schemaIncludes>
                    <bindingDirectory>${project.basedir}/src/main/xsd</bindingDirectory>
                    <bindingIncludes>
                        <include>global.xjb</include>
                    </bindingIncludes>
                    <generateDirectory>${project.basedir}/src/generated</generateDirectory>
                </configuration>
            </plugin>
```

Let's see how we can do marshalling and unmarshalling.

Marshalling means converting java objects to xml and unmarshalling means converting xml back to java objects.

JAXBDemo.java

```java
package com.praveen.jaxb;

import java.io.StringReader;


public class JAXBDemo {

    public static void main(String[] args) {

        try {
            JAXBContext context = JAXBContext.newInstance(Patient.class);
            Marshaller marshaller = context.createMarshaller();

            Patient patient = new Patient();
            patient.setId(1);
            patient.setName("Praveen Oruganti");

            StringWriter writer = new StringWriter();
            marshaller.marshal(patient, writer);

            System.out.println(writer.toString());

            Unmarshaller unMarshaller = context.createUnmarshaller();

            Patient patientResult = (Patient) unMarshaller.unmarshal(new StringReader(writer.toString()));
            System.out.print(patientResult.getName());

        } catch (JAXBException e) {
            e.printStackTrace();
        }
    }

}
```

Output:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><patient id="1" xmlns="http://praveenoruganti.blogspot.com/Patient"><name>Praveen Oruganti</name><age>0</age></patient>
Praveen Oruganti

You can view this code in my repository
(https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-jaxb-xjc-master)

JAXB provides runtime annotations i.e...

- ✓ @XMLRoot
- ✓ @XMLElement
- ✓ @XMLAttribute

Apache CXF

Apache CXF is a JAX-WS fully compliant framework and a webservice engine.
On top of features defined by JAX-WS standards, Apache CXF provides the capability of conversion between WSDL and Java classes, APIs used to manipulate raw XML messages, the support for JAX-RS, integration with the Spring Framework, etc.

Why CXF?

- ✓ **SOAP/REST engine**
  - ➢ Serialize and De- serialize
  - ➢ Publish and Dispatch
- ✓ XML -> Soap/Rest endpoint -> Java Object -> WS Endpoint Method and it follows the same route while sending back the response.
- ✓ **Webservice Standards**
  - ➢ WS-Security
  - ➢ WS-Policy etch

- ✓ **It also provides tools like**
  - ➢ Wsdl2Java
  - ➢ Java2Wsdl
- ✓ It is very easy to configure as it uses spring configuration
- ✓ It is also easy to extend/customize using Interceptors and Handlers

Let's start coding Apache CXF JAX WS using below designs

1. Bottom Up Approach

```
<dependencies>
	<!-- apache cxf jax-ws 3.0.2 -->
	<dependency>
		<groupId>org.apache.cxf</groupId>
		<artifactId>cxf-rt-frontend-jaxws</artifactId>
		<version>3.0.2</version>
		<scope>compile</scope>
	</dependency>
	<dependency>
		<groupId>org.apache.cxf</groupId>
		<artifactId>cxf-rt-transports-http</artifactId>
		<version>3.0.2</version>
		<scope>compile</scope>
	</dependency>

	<!-- spring framework 4.1.0 -->
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-web</artifactId>
		<version>4.1.0.RELEASE</version>
		<scope>compile</scope>
	</dependency>
</dependencies>
```

apache-cxf-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:jaxrs="http://cxf.apache.org/jaxrs" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

    <jaxws:endpoint id="bookservice"
        implementor="com.praveen.apachecx.soap.service.BookServiceImpl"
        address="/book">
    </jaxws:endpoint>

    <jaxws:endpoint id="paymentService"
        implementor="com.praveen.apachecx.soap.service.PaymentProcessorImpl"
        address="/paymentProcessor">
    </jaxws:endpoint>

</beans>
```

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" 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">

    <display-name>praveenoruganti-apachecxf-jaxws-bottom-up</display-name>

    <!-- Apache CXF -->
    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>

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

    <!-- web context param -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>WEB-INF/apache-cxf-services.xml</param-value>
    </context-param>

    <!-- session timeout -->
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
```

IBookService.java

```java
package com.praveen.apachecx.soap.service;

import javax.jws.WebService;

@WebService(name="BookService")
@SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
public interface IBookService {

    public String getBookNameBasedOnISBN(String isbnNumber);
}
```

BookServiceImpl.java

```java
package com.praveen.apachecx.soap.service;

import javax.jws.WebService;

@WebService(serviceName="BookService", endpointInterface="com.praveen.apachecx.soap.service.IBookService")
public class BookServiceImpl implements IBookService {

    public String getBookNameBasedOnISBN(String isbnNumber) {

        if(isbnNumber.equalsIgnoreCase("ISBN-2134")){
            return "Microbiology";
        }
        return "Invalid_ISBN_Number";
    }
}
```

PaymentProcessor.java

```java
package com.praveen.apachecx.soap.service;

import javax.jws.WebParam;

@WebService(serviceName = "PaymentProcessor")
public interface PaymentProcessor {

    public @WebResult(name = "response")
    PaymentProcessorResponse processPayment(
            @WebParam(name = "PaymentProcessorRequest") PaymentProcessorRequest paymentProcessorRequest);
}
```

PaymentProcessorImpl.java

```java
package com.praveen.apachecx.soap.service;

import com.praveen.apachecx.soap.service.dto.PaymentProcessorRequest;

public class PaymentProcessorImpl implements PaymentProcessor {

    public PaymentProcessorResponse processPayment(
            PaymentProcessorRequest paymentProcessorRequest) {
        PaymentProcessorResponse paymentProcessorResponse = new PaymentProcessorResponse();
        //Business Logic or a call  to a Business Logic Class Goes Here.
        paymentProcessorResponse.setResult(true);
        return paymentProcessorResponse;
    }

}
```

CreditCardInfo.java

```java
package com.praveen.apachecx.soap.dto;

import java.util.Date;

@XmlType(name = "CreditCardInfo")
@XmlAccessorType(XmlAccessType.FIELD)
public class CreditCardInfo {

    @XmlElement(name = "cardNumber", required = true)
    String cardNumber;
    @XmlElement(name = "expirtyDate", required = true)
    private Date expirtyDate;
    @XmlElement(name = "firstName", required = true)
    String firstName;
    @XmlElement(name = "lastName", required = true)
    String lastName;
    @XmlElement(name = "secCode", required = true)
    String secCode;
    @XmlElement(name = "Address", required = true)
    String Address;

    public String getCardNumber() {
        return cardNumber;
    }

    public void setCardNumber(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
```

PaymentProcessorRequest.java

```java
package com.praveen.apachecx.soap.dto;

import javax.xml.bind.annotation.XmlAccessType;

@XmlType(name = "PaymentProcessorRequest")
@XmlAccessorType(XmlAccessType.FIELD)
public class PaymentProcessorRequest {

    @XmlElement(name = "creditCardInfo", required = true)
    private CreditCardInfo creditCardInfo;
    private Double amount;

    public CreditCardInfo getCreditCardInfo() {
        return creditCardInfo;
    }

    public void setCreditCardInfo(CreditCardInfo creditCardInfo) {
        this.creditCardInfo = creditCardInfo;
    }

    public Double getAmount() {
        return amount;
    }

    public void setAmount(Double amount) {
        this.amount = amount;
    }

}
```

PaymentProcessorResponse.java

```java
package com.praveen.apachecx.soap.dto;

import javax.xml.bind.annotation.XmlType;

@XmlType(name = "PaymentProcessorResponse")
public class PaymentProcessorResponse {

    private boolean result;

    public boolean isResult() {
        return result;
    }

    public void setResult(boolean result) {
        this.result = result;
    }
}
```

Praveen Oruganti Apache CXF: JAX-WS-Bottom-UP

http://localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/

Praveen Oruganti Apache CXF: JAX-WS : Bottom UP approach

SOAP Web Service

CXF - Service list

localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services

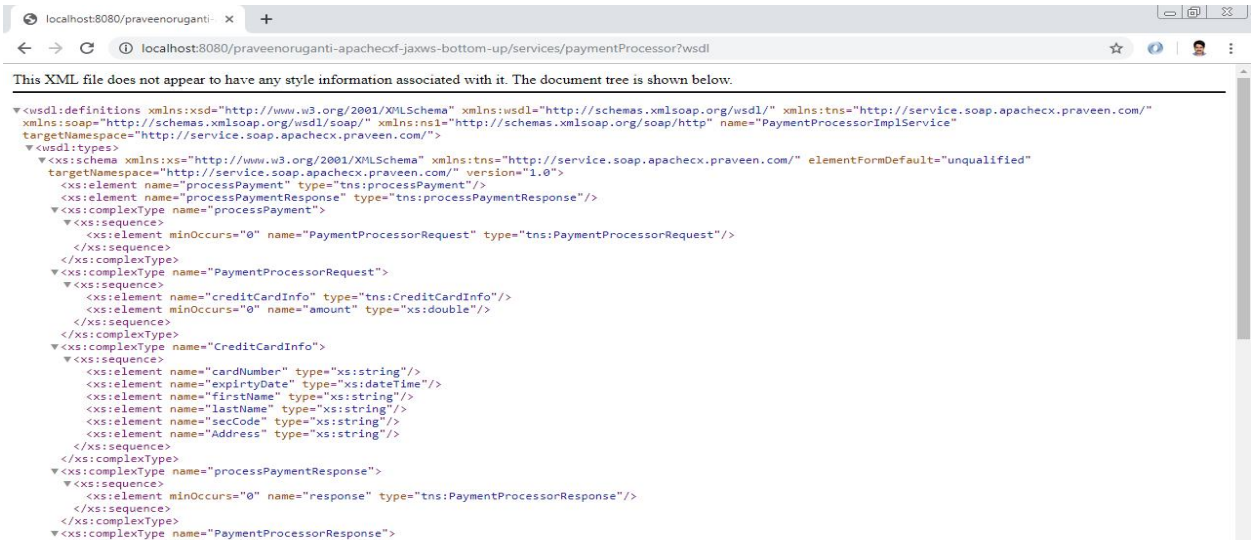
Available SOAP services:

BookService • getBookNameBasedOnISBN	**Endpoint address:** http://localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services/book **WSDL :** {http://service.soap.apachecx.praveen.com/}BookService **Target namespace:** http://service.soap.apachecx.praveen.com/
PaymentProcessor • processPayment	**Endpoint address:** http://localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services/paymentProcessor **WSDL :** {http://service.soap.apachecx.praveen.com/}PaymentProcessorImplService **Target namespace:** http://service.soap.apachecx.praveen.com/

localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services/book?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://service.soap.apachecx.praveen.com/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="BookService"
 targetNamespace="http://service.soap.apachecx.praveen.com/">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://service.soap.apachecx.praveen.com/" elementFormDefault="unqualified"
     targetNamespace="http://service.soap.apachecx.praveen.com/" version="1.0">
      <xs:element name="getBookNameBasedOnISBN" type="tns:getBookNameBasedOnISBN"/>
      <xs:element name="getBookNameBasedOnISBNResponse" type="tns:getBookNameBasedOnISBNResponse"/>
      <xs:complexType name="getBookNameBasedOnISBN">
        <xs:sequence>
          <xs:element minOccurs="0" name="arg0" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="getBookNameBasedOnISBNResponse">
        <xs:sequence>
          <xs:element minOccurs="0" name="return" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getBookNameBasedOnISBNResponse">
    <wsdl:part element="tns:getBookNameBasedOnISBNResponse" name="parameters"> </wsdl:part>
  </wsdl:message>
  <wsdl:message name="getBookNameBasedOnISBN">
    <wsdl:part element="tns:getBookNameBasedOnISBN" name="parameters"> </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="BookService">
    <wsdl:operation name="getBookNameBasedOnISBN">
      <wsdl:input message="tns:getBookNameBasedOnISBN" name="getBookNameBasedOnISBN"> </wsdl:input>
      <wsdl:output message="tns:getBookNameBasedOnISBNResponse" name="getBookNameBasedOnISBNResponse"> </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BookServiceSoapBinding" type="tns:BookService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getBookNameBasedOnISBN">
      <soap:operation soapAction="" style="document"/>
```



You can view this code in my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-apachecxf-jaxws-bottom-up-master)

2. Top Down Approach

```
            <plugin>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-codegen-plugin</artifactId>
                <version>3.0.2</version>
                <executions>
                    <execution>
                        <configuration>
                            <sourceRoot>generated/java/source</sourceRoot>
                            <wsdlOptions>
                                <wsdlOption>
                                    <wsdl>src/main/resources/com/praveen/apachecxf/soap/service/BookService.wsdl</wsdl>
                                </wsdlOption>
                                <wsdlOption>
                                    <wsdl>src/main/resources/com/praveen/apachecxf/soap/service/paymentProcessor.wsdl</wsdl>
                                </wsdlOption>
                            </wsdlOptions>
                        </configuration>
                        <goals>
                            <goal>wsdl2java</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

```
        <dependencies>
            <!-- apache cxf jax-ws 3.0.2 -->
            <dependency>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-rt-frontend-jaxws</artifactId>
                <version>3.0.2</version>
                <scope>compile</scope>
            </dependency>
            <dependency>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-rt-transports-http</artifactId>
                <version>3.0.2</version>
                <scope>compile</scope>
            </dependency>

            <!-- spring framework 4.1.0 -->
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-web</artifactId>
                <version>4.1.0.RELEASE</version>
                <scope>compile</scope>
            </dependency>
        </dependencies>
```

BookServiceImpl.java

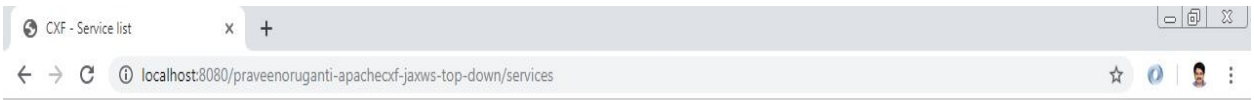
```java
package com.praveen.apachecxf.soap.service;

import javax.jws.WebService;

@WebService(serviceName="BookService", endpointInterface="com.praveen.apachecxf.soap.service.BookService")
public class BookServiceImpl implements BookService {

    public String getBookNameBasedOnISBN(String isbnNumber) {

        if(isbnNumber.equalsIgnoreCase("ISBN-2134")){
            return "Microbiology";
        }
        return "Invalid_ISBN_Number";
    }
}
```

PaymentProcessorImpl.java

```java
package com.praveen.apachecxf.soap.service;

public class PaymentProcessorImpl implements PaymentProcessor {

    public PaymentProcessorResponse processPayment(
            PaymentProcessorRequest paymentProcessorRequest) {
        PaymentProcessorResponse paymentProcessorResponse = new PaymentProcessorResponse();
        //Business Logic or a call  to a Business Logic Class Goes Here.
        paymentProcessorResponse.setResult(true);
        return paymentProcessorResponse;
    }

}
```

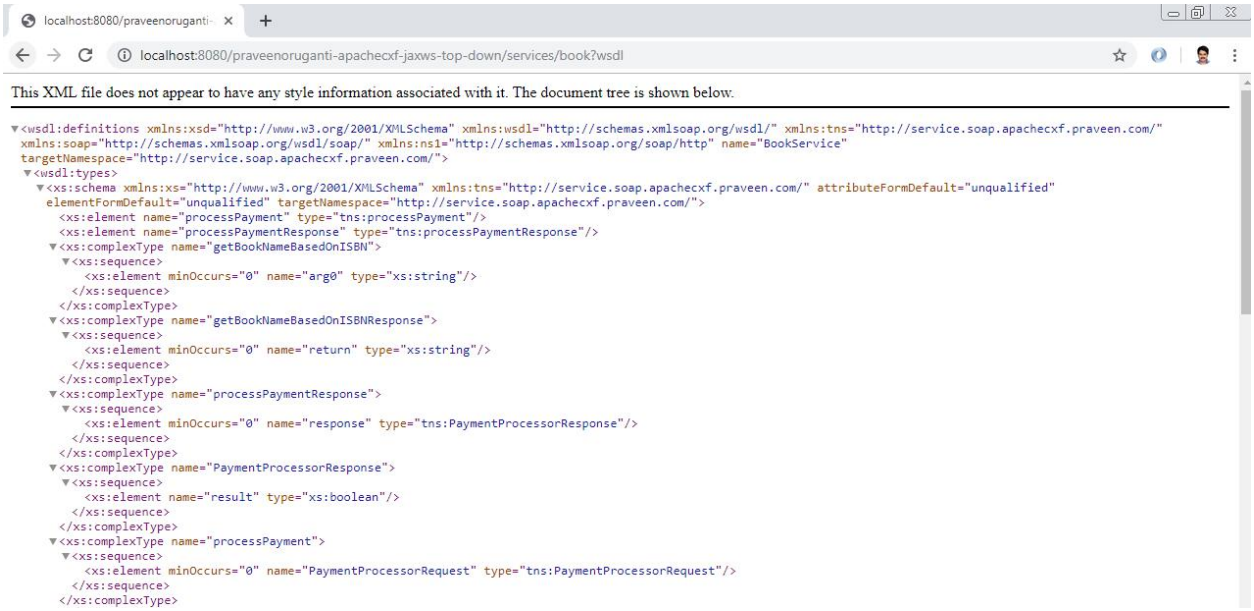
Praveen Oruganti Apache CXF: JAX-WS : Top Down approach

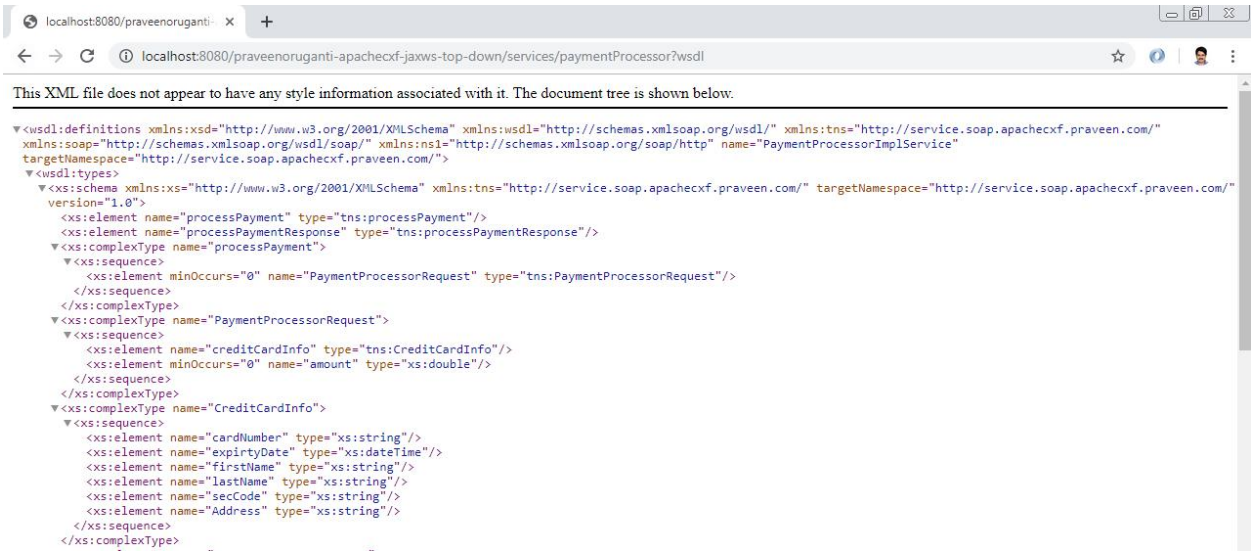
SOAP Web Service



Available SOAP services:

BookService • getBookNameBasedOnISBN	**Endpoint address:** http://localhost:8080/praveenoruganti-apachecxf-jaxws-top-down/services/book **WSDL :** {http://service.soap.apachecxf.praveen.com/}BookService **Target namespace:** http://service.soap.apachecxf.praveen.com/
PaymentProcessor • processPayment	**Endpoint address:** http://localhost:8080/praveenoruganti-apachecxf-jaxws-top-down/services/paymentProcessor **WSDL :** {http://service.soap.apachecxf.praveen.com/}PaymentProcessorImplService **Target namespace:** http://service.soap.apachecxf.praveen.com/





SOAPClient.java

```
1  package com.praveen.soap.client;
2
3  import java.io.IOException;
4
5  public class SOAPClient {
6
7      /**
8       * wsimport - JAX-WS top-down web service approach main() method to test/start
9       * soap web service
10      *
11      * @param args
12      * @throws IOException
13      */
14     public static void main(String[] args) throws IOException {
15
16         //String httpRequestURL = "http://localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services/book/BookService?wsdl";
17         String httpRequestURL = "http://localhost:8080/praveenoruganti-apachecxf-jaxws-top-down/services/book/BookService?wsdl";
18         String requestXmlParam = "<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:ser=\"http://service.soap.apachecxf.praveen
19                 + " <soapenv:Header/>" + "<soapenv:Body>" + "<ser:getBookNameBasedOnISBN>" + "<arg0>ISBN-2134</arg0>"
20                 + " </ser:getBookNameBasedOnISBN>" + "</soapenv:Body>" + "</soapenv:Envelope>";
21         System.out.println(httpRequestURL);
22         String responseString = ClientUtils.testService(httpRequestURL, requestXmlParam);
23         System.out.println("Response String : " + responseString);
24     }
25
26 }
```

Output

http://localhost:8080/praveenoruganti-apachecxf-jaxws-top-down/services/book/BookService?wsdl
Response code: 200
ResponseMessageFromServer: null
Response String : <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:getBookNameBasedOnISBNResponse xmlns:ns2="http://service.soap.apachecxf.praveen.com/"><return>Microbiology</return></ns2:getBookNameBasedOnISBNResponse></soap:Body></soap:Envelope>

Please note: Top Down is the preferred approach.

You can view this code in my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-apachecxf-jaxws-top-down-master)

Adding WS-Security policy using UsernameToken profile

There are two options available to protect the exposed web service

- ✓ Adding WS-Security policy to WSDL document
- ✓ Implementing CXF interceptors (policy-free WSDL)

Implementing CXF interceptors

apache-cxf-services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:jaxrs="http://cxf.apache.org/jaxrs" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

    <jaxws:endpoint id="bookservice"
        implementor="com.praveen.apachecxf.soap.service.BookServiceImpl"
        address="/book">
        <jaxws:inInterceptors>
            <bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
                <constructor-arg>
                    <map>
                        <entry key="action" value="UsernameToken" />
                        <entry key="passwordType" value="PasswordText" />
                        <entry key="passwordCallbackRef" value-ref="myPasswordCallback" />
                    </map>
                </constructor-arg>
            </bean>
        </jaxws:inInterceptors>
    </jaxws:endpoint>
```

UTPasswordCallback.java

```java

package com.praveen.apachecxf.soap.security;

import java.io.IOException;

public class UTPasswordCallback implements CallbackHandler {

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

    public UTPasswordCallback() {

        passwords.put("Praveen", "Praveen");
        passwords.put("cxf", "cxf");
    }

    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];

            String pass = passwords.get(pc.getIdentifier());
            if (pass != null) {
                pc.setPassword(pass);
                return;
            }
        }
    }

}
```

If you see above code, we have implemented the below

- ✓ Implement the call back handler class that can return the password for the given user id.
- ✓ Configure the User Name token profile security in the cxf-servlet.xml

How to exchange files using MTOM?

- ✓ Use the DataHandler type as the bean property or parameters to the web service method.
- ✓ Enable the MTOM property on the endpoint in the cxf-servlet.xml

Soap Client:

You can use SOAPUI for testing or you can use the below client program for testing soap based webservices.

ClientUtils.java

```java
package com.praveen.soap.client;

import java.io.BufferedReader;

public class ClientUtils {
    /**
     * This method uses HttpURLConnection to invoke exposed SOAP web service and returns the response string to the calling client
     *
     * @param httpRequestURL
     * @param requestXmlParam
     * @return responseXML
     * @throws IOException
     */
    public static String testService(String httpRequestURL, String requestXmlParam) throws IOException {

        // local variables
        URL url = null;
        HttpURLConnection httpURLConnection = null;
        OutputStreamWriter outputStreamWriter = null;
        String responseMessageFromServer = null;
        String responseXML = null;

        try {
            // set basic request parameters
            url = new URL(httpRequestURL);
            httpURLConnection = (HttpURLConnection) url.openConnection();
            httpURLConnection.setDoOutput(true);
            httpURLConnection.setRequestMethod("POST");
            //          httpURLConnection.setRequestProperty("SOAPAction", "");
            httpURLConnection.setRequestProperty("Content-Type", "text/xml");
            httpURLConnection.setRequestProperty("Accept", "application/xml");

            // write request XML to the HTTP request
            outputStreamWriter = new OutputStreamWriter(httpURLConnection.getOutputStream());
            outputStreamWriter.write(requestXmlParam);
            outputStreamWriter.flush();

            System.out.println("Response code: " + httpURLConnection.getResponseCode());

            if (httpURLConnection.getResponseCode() == 200) {

                responseMessageFromServer = httpURLConnection.getResponseMessage();
                System.out.println("ResponseMessageFromServer: " + responseMessageFromServer);
                responseXML = getResponseXML(httpURLConnection);
            }
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
            throw ioex;
        }
        finally{
            // finally close all operations
            outputStreamWriter.close();
            httpURLConnection.disconnect();
        }
        return responseXML;
    }
```

ClientUtils.java

```java
    /**
     * This method is used to get response XML from the HTTP POST request
     *
     * @param httpURLConnection
     * @return stringBuffer.toString()
     * @throws IOException
     */
    private static String getResponseXML(HttpURLConnection httpURLConnection) throws IOException {

        // local variables
        StringBuffer stringBuffer = new StringBuffer();
        BufferedReader bufferedReader = null;
        InputStreamReader inputStreamReader = null;
        String readSingleLine = null;

        try{
            // read the response stream AND buffer the result into a StringBuffer
            inputStreamReader = new InputStreamReader(httpURLConnection.getInputStream());
            bufferedReader = new BufferedReader(inputStreamReader);

            // reading the XML response content line BY line
            while ((readSingleLine = bufferedReader.readLine()) != null) {
                stringBuffer.append(readSingleLine);
            }
        }
        catch (IOException ioex) {
            ioex.printStackTrace();
            throw ioex;
        }
        finally{
            // finally close all operations
            bufferedReader.close();
            httpURLConnection.disconnect();
```

SOAPClient.java

```java
package com.praveen.soap.client;

import java.io.IOException;

public class SOAPClient {

    /**
     * wsimport - JAX-WS top-down web service approach main() method to test/start
     * soap web service
     *
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {

        //String httpRequestURL = "http://localhost:8080/praveenoruganti-apachecxf-jaxws-bottom-up/services/book/BookService?wsdl";
        String httpRequestURL = "http://localhost:8080/praveenoruganti-apachecxf-jaxws-top-down/services/book/BookService?wsdl";
        String requestXmlParam = "<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:ser=\"http://service.soap.apachecxf.praveen
                + " <soapenv:Header/>" + "<soapenv:Body>" + "<ser:getBookNameBasedOnISBN>" + "<arg0>ISBN-2134</arg0>"
                + " </ser:getBookNameBasedOnISBN>" + "</soapenv:Body>" + "</soapenv:Envelope>";
        System.out.println(httpRequestURL);
        String responseString = ClientUtils.testService(httpRequestURL, requestXmlParam);
        System.out.println("Response String : " + responseString);
    }

}
```

Let's implement soap webservice using SpringBoot

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-services</artifactId>
</dependency>
```

```xml
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>jaxb2-maven-plugin</artifactId>
                <version>1.6</version>
                <executions>
                    <execution>
                        <id>xjc</id>
                        <goals>
                            <goal>xjc</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <schemaDirectory>${project.basedir}/src/main/resources</schemaDirectory>
                    <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
                    <clearOutputDir>false</clearOutputDir>
                </configuration>
            </plugin>
            <!-- JAXB2 Maven Plugin -->
            <!-- XSD Source Folder -->
            <!-- Java Class Source Folder -->
            <!-- clear folder -> false -->

        </plugins>
    </build>
```

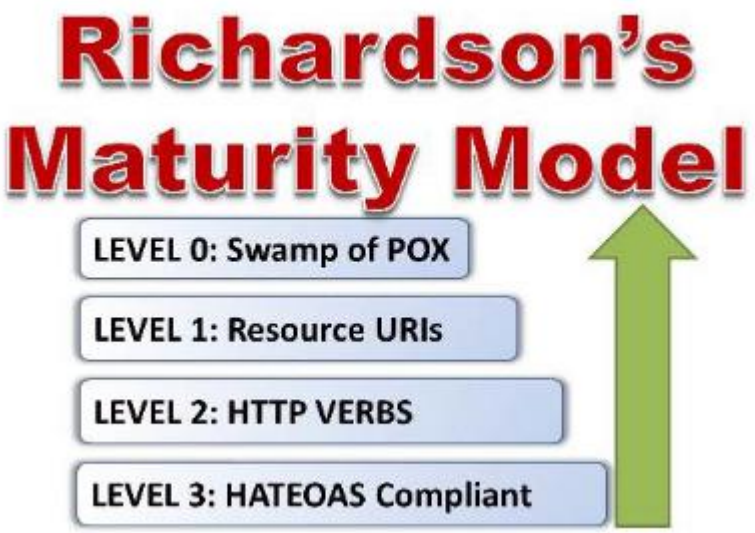
You can get the complete code for this from my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-soapws-springboot-master)

What is REST?

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation.
Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful. These principles are listed below.

- ✓ REST stands for Representational State Transfer
- ✓ Restful webservices is a stateless client-server architecture where webservices are resources and can be identified by their URIs.
- ✓ REST Client applications can use HTTP GET/POST/PUT/DELETE methods invoke Resetful Webservice.
- ✓ Lightweight and doesn't follow any standards unlike SOAP Webservices.

For REST APIs we need to follow Richardson's Maturity Model



So, how do you measure if a web service is REST web service or not?

This can be done using a model developed by a Computer Scientist called Roy Fielding. This model is called Richardson's Maturity Model and is use to determine how much the API conforms the certain architectural styles.

It is used to answer the question: How RESTful is the API? Based on certain criteria, a REST API can can score from 0 – 3 ranging from Not-RESTful (level 0) to Fully-RESTful(level 3)

Level 0: Not RESTful

At level 0 a REST API is described as not RESTful. This means that the API is made up just XML and therefore known as 'Swamp of POX', plain old xml. At level 0, the request and response uses just on entry point URI and just one method HTTP POST method. Level 0 is equivalent to SOAP(Simple Object Access Protocol) web service

Level 1: Resource-based URIs

Level 1 has to do with Resource and Resource-based URIs. This means that there are different URIs for different resources in the API. URI stands for Universal Resource Identifier. It is similar to a URL. The difference is that a URI is a URL that maps to a particular resource.
Resources are endpoints in the API which could represent an object, collections of objects, operations on objects etc.

Take example of URL from a messaging application:

http://praveenoruganti.blogspot.com/messages
http://praveenoruganti.blogspot.com/messages/1
http://praveenoruganti.blogspot.com/messages/2/delete

The first line is a URI that maps to a collection of messages and a returns a list of messages to the client.
The second line maps to a single message with ID of 1 and sends back the message with the id of 1 to the client.
The third URI maps to the delete operation for message with ID of 2 and allows the user to delete the message
So if your REST API has the URIs designed this way, then that API have attained to REST level 1.

Level 2: HTTP Methods (or Verbs)

If a REST API implements resource-based URIs and also uses the appropriate HTTP verb to perform operations, then the API have attained REST level 2.
The HTTP verbs corresponds to the four CRUD operations that can be performed on a resource. The verbs are:

GET: used to retrieve resource(s) from the server
POST: used to create new resource
PUT: used to modify existing resource
DELETE: used to delete a resource from the server
So if a client makes a request to retrieve a resource from the server, the it should use the REST the GET method.
The fact is that without the use of HTTP verbs, the API would still work but would not be fully RESTful.

Level 3: HATEOAS Compliant

To attain level 3 after attaining level 2, then the API must be HATEOAS compliant. HATEOAS stands for Hypermedia as the Engine of Application State. Of course you know what Hypermedia is. Typical example is hyperlinks. So this level requires that the API must provide in addition to the response, links to other resources or operations that the client may need.
So if a client request for a resource, say a user profile, then the API should provide the user with links to Edit, Delete and Save as well as link to get list of users.

This level is what makes is possible to browse a website without much help because the hyperlinks and images provide the user with hints on how to move to a different part of the application (changing the application state)

REST Principles

1.Client-Server

- ✓ The constraint states that a REST application should have a Client Server Architecture.
- ✓ Advantage is Client and Server are seperated
- ✓ They can evolve independently
- ✓ Clients need not know anything about business logic/data access layer
- ✓ Server need not know anything about front end UI

2. Stateless

- ✓ Stateless constraint states that the server does not store any session data.
- ✓ The communication between client and server is stateless.
- ✓ It means all the information to understand a request is contained within request.
- ✓ Improves Scalability.

3. Cacheable

- ✓ Cacheable constraint states responses should be cacheable, if possible.
- ✓ It requires every response should include whether a response is cacheable or not.
- ✓ For subsequent requests, the client can retrieve from its cache, need to send request to server.
- ✓ Reduces network latency.

4. Uniform interface

Uniform interface is the key differentiation between REST APIs and NON REST APIs. There are 4 elements of Uniform interface constraint

- ✓ Identification of Resources(typically by an URL)
- ✓ Manipulation of Resources through representations.
- ✓ Self descriptive messages for each request.
- ✓ HATEOS(Hypermedia As The Engine Of application State)
- ✓ Promotes generally as all components interact in the same way

5. Layered System

- ✓ Allows an architecture to be composed of hierarchical layers.
- ✓ Each layer don't know anything beyond the immediate layer.
- ✓ Limits the amount of complexity that can be introduced at any single layer.
- ✓ Disadvantage is latency.

6. Code on Demand

- ✓ Optional Constraint.
- ✓ In addition to data,the server can provide executable code to the client.
- ✓ This constraint reduces the visibility.

SOAP vs REST Web Services

- ✓ SOAP is a Protocol. Whereas Rest is an architectural style.
- ✓ SOAP stands for Simple Object Access Protocol. Whereas Rest stands for REpresentational State Transfer.
- ✓ SOAP can't use REST because it is a protocol. Whereas REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
- ✓ SOAP uses services interfaces to expose the business logic. Whereas REST uses URI to expose business logic.
- ✓ JAX-WS is the java API for SOAP web services. Whereas JAX-RS is the java API for RESTful web services.
- ✓ SOAP defines standards to be strictly followed. Whereas REST doesn't define too much standards like SOAP.
- ✓ SOAP requires more bandwidth and resource than REST. Whereas REST requires less bandwidth and resource than SOAP.
- ✓ SOAP defines its own security. Whereas RESTful web services inherits security measures from the underlying transport.
- ✓ SOAP permits XML data format only. Whereas REST permits different data format such as Plain text, HTML, XML, JSON etc.
- ✓ SOAP is less perferred than REST. Whereas REST is more preferred than SOAP.

HTTP Methods

GET

- ✓ HTTP GET method is used to **read** (or retrieve) a representation of a resource.

- ✓ According to the design of the HTTP specification, GET requests are used only to read data and not change it.
- ✓ Therefore, when used this way, they are considered safe.
- ✓ That is, they can be called without risk of data modification or corruption. Means calling it once has the same effect as calling it 10 times.
- ✓ Additionally, GET is idempotent which means that making multiple identical requests ends up having the same result as a single request.
- ✓ Donot expose unsafe operations via GET. It should never modify any resources on the server.

Example:
GET http://www.example.com/customers/12345/orders

POST

- ✓ The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources.
- ✓ That is, subordinate to some other (e.g. parent) resource.
- ✓ In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI), etc.
- ✓ On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.
- ✓ POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.

Example:
POST http://www.example.com/customers/12345/orders

PUT

- ✓ PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.
- ✓ However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID.
- ✓ Again, the request body contains a resource representation. Many feel this is convoluted and confusing.Consequently, this method of creation should be used sparingly, if at all.
- ✓ Alternatively, use POST to create new resources and provide the client-defined ID in the body representation- resumably to a URI that doesn't include the ID of the resource (see POST below).
- ✓ On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. A body in the response is optional- providing one consumes more

bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID.

- ✓ PUT is not a safe operation, in that it modifies (or creates) state on the server, but it is idempotent. In other words, if you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.
- ✓ If, for instance, calling PUT on a resource increments a counter within the resource, the call is no longer idempotent. Sometimes that happens and it may be enough to document that the call is not idempotent. However, it's recommended to keep PUT requests idempotent. It is strongly recommended to use POST for non-idempotent requests.

Example:
PUT http://www.example.com/customers/12345/orders/98765

PATCH

- ✓ PATCH is used for **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource.
- ✓ This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version.
- ✓ This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch.
- ✓ PATCH is neither safe nor idempotent. However, a PATCH request can be issued in such a way as to be idempotent, which also helps prevent bad outcomes from collisions between two PATCH requests on the same resource in a similar time frame.
- ✓ Collisions from multiple PATCH requests may be more dangerous than PUT collisions because some patch formats need to operate from a known base-point or else they will corrupt the resource.
- ✓ Clients using this kind of patch application should use a conditional request such that the request will fail if the resource has been updated since the client last accessed the resource.

For example, the client can use a strong ETag in an If-Match header on the PATCH request.

Example:
PATCH http://www.example.com/customers/12345/orders/98765

DELETE

- ✓ DELETE is pretty easy to understand. It is used to **delete** a resource identified by a URI.

- ✓ On successful deletion, return HTTP status 200 (OK) along with a response body, perhaps the representation of the deleted item (often demands too much bandwidth), or a wrapped response (see Return Values below).Either that or return HTTP status 204 (NO CONTENT) with no response body. In other words, a 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses.
- ✓ HTTP-spec-wise, DELETE operations are idempotent. If you DELETE a resource, it's removed. Repeatedly calling DELETE on that resource ends up the same: the resource is gone. If calling DELETE say, decrements a counter (within the resource), the DELETE call is no longer idempotent. As mentioned previously, usage statistics and measurements may be updated while still considering the service idempotent as long as no resource data is changed. Using POST for non-idempotent resource requests is recommended.
- ✓ There is a caveat about DELETE idempotence, however. Calling DELETE on a resource a second time will often return a 404 (NOT FOUND) since it was already removed and therefore is no longer findable. This, by some opinions, makes DELETE operations no longer idempotent, however, the end-state of the resource is the same. Returning a 404 is acceptable and communicates accurately the status of the call.

Example:
DELETE http://www.example.com/customers/12345/orders

What is the difference between HTTP POST and PUT requests?

- ✓ The HTTP methods POST and PUT aren't the HTTP equivalent of the CRUD's create and update. They both serve a different purpose. It's quite possible, valid and even preferred in some occasions, to use POST to create resources, or use PUT to update resources.
- ✓ Use PUT when you can update a resource completely through a specific resource. If you do not know the actual resource location, for instance, when you add a new object, but do not have any idea where to store it, you can POST it to an URL, and let the server decide the actual URL.
- ✓ As soon as you know the new resource location, you can use PUT again to do updates to the blue stapler article. But as said before: you CAN add new resources through PUT as well. The next example is perfectly valid if your API provides this functionality PUT and POST are both unsafe methods. However, PUT is idempotent, while POST is not

Here with the Http status codes

1×× Informational
100 Continue
101 Switching Protocols
102 Processing
2×× Success

200 OK
201 Created
202 Accepted
203 Non-authoritative Information
204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status
208 Already Reported
226 IM Used

3×× Redirection

300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
308 Permanent Redirect

4×× Client Error

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot
421 Misdirected Request
422 Unprocessable Entity
423 Locked
424 Failed Dependency
426 Upgrade Required
428 Precondition Required

429 Too Many Requests
431 Request Header Fields Too Large
444 Connection Closed Without Response
451 Unavailable For Legal Reasons
499 Client Closed Request
5×× Server Error
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected
510 Not Extended
511 Network Authentication Required
599 Network Connect Timeout Error

What kind of output formats can one generate using RESTful web service?

Xml, JSON, TEXT ETC

Now let's see how we can build REST Webservice using Apache CXF JAX-RS

We need to add below dependencies in pom.xml

```
<dependencies>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>3.0.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>3.0.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-rs-service-description</artifactId>
            <version>3.0.0-milestone1</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.2.0.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.0.RELEASE</version>
        </dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-jaxrs</artifactId>
    <version>1.9.13</version>
</dependency>
```

web.xml

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

<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="2.5"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name>cxf</display-name>
    <servlet>
        <description>Apache CXF Endpoint</description>
        <display-name>cxf</display-name>
        <servlet-name>cxf</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxf</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
</web-app>
```

cxf-servlet.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:soap="http://cxf.apache.org/bindings/soap"
    xmlns:jaxrs="http://cxf.apache.org/jaxrs"
    xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://cxf.apache.org/bindings
    <jaxrs:server id="patientService"
        address="/patientservice">
        <jaxrs:serviceBeans>
            <ref bean="serviceBean" />
        </jaxrs:serviceBeans>

        <jaxrs:providers>
            <ref bean="jacksonProvider" />
        </jaxrs:providers>
    </jaxrs:server>

    <bean id="serviceBean"
        class="com.praveen.apachecxf.rest.PatientServiceImpl"></bean>

    <bean id="jacksonProvider"
        class="org.codehaus.jackson.jaxrs.JacksonJsonProvider" />
</beans>
```

PatientService.java

```java
package com.praveen.apachecxf.rest;

import javax.ws.rs.DELETE;

@Produces({ "application/xml", "application/json" })
public interface PatientService {

    @GET
    @Path("/patients/{id}/")
    Patient getPatient(@PathParam("id") String id);

    @PUT
    @Path("/patients/")
    Response updatePatient(Patient patient);

    @POST
    @Path("/patients/")
    Response addPatient(Patient patient);

    @DELETE
    @Path("/patients/{id}/")
    Response deletePatients(@PathParam("id") String id);

    @Path("/prescriptions/{id}")
    Prescription getPrescription(@PathParam("id") String prescriptionId);

}
```

Medicine.java

```java
package com.praveen.apachecxf.rest;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "Medicine")
public class Medicine {
    private long id;
    private String description;

    public long getId() {
        return id;
    }

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

    public String getDescription() {
        return description;
    }

    public void setDescription(String d) {
        this.description = d;
    }
}
```

Patient.java

```java
package com.praveen.apachecxf.rest;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "Patient")
public class Patient {
    private long id;
    private String name;

    public Patient() {

    }

    public long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

Prescription.java

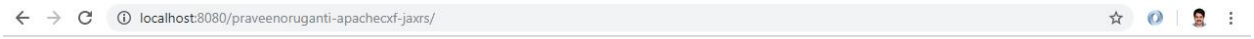
```java
package com.praveen.apachecxf.rest;

import java.util.HashMap;

@XmlRootElement(name = "Prescription")
public class Prescription {
    private long id;
    private String description;
    private Map<Long, Medicine> prescriptions = new HashMap<Long, Medicine>();

    public Prescription() {
        init();
    }

    public long getId() {
        return id;
    }

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

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @GET
    @Path("/medicines/{id}")
    public Medicine getMedicine(@PathParam("id")String medicineid) {
        System.out.println("----invoking getMedicine with id: " + medicineid);
```

```
        Medicine medicine = prescriptions.get(new Long(medicineid));
        return medicine;
    }

    final void init() {
        Medicine medicine = new Medicine();
        medicine.setId(323);
        medicine.setDescription("Medicine 323");
        prescriptions.put(medicine.getId(), medicine);
    }
}
```

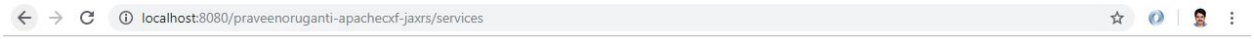
PatientServiceImpl.java

```
package com.praveen.apachecxf.rest;

import java.util.HashMap;

public class PatientServiceImpl implements PatientService {
    private long currentId = 123;
    Map<Long, Patient> patients = new HashMap<Long, Patient>();
    Map<Long, Prescription> prescriptions = new HashMap<Long, Prescription>();

    public PatientServiceImpl() {
        init();
    }

    final void init() {
        Patient patient = new Patient();
        patient.setName("John");
        patient.setId(currentId);
        patients.put(patient.getId(), patient);

        Prescription prescription = new Prescription();
        prescription.setDescription("prescription 223");
        prescription.setId(223);
        prescriptions.put(prescription.getId(), prescription);
    }

    public Response addPatient(Patient patient) {
        System.out.println("...invoking addPatient, Patient Name is... " + patient.getName());
        patient.setId(++currentId);
        patients.put(patient.getId(), patient);
        return Response.ok(patient).build();
    }

    public Patient getPatient(String id) {
        System.out.println("...invoking getPatient, Patient Id is... " + id);
        Long patientId = Long.parseLong(id);
        Patient patient = patients.get(patientId);
        return patient;
    }

    public Response updatePatient(Patient updatedPatient) {
        System.out.println("...invoking updatePatient, Patient Name is... " + updatedPatient.getName());

        Patient currentPatient = patients.get(updatedPatient.getId());

        Response response = null;
        if (currentPatient != null) {
            patients.put(updatedPatient.getId(), updatedPatient);
            response = Response.ok().build();
        } else {
            response = Response.notModified().build();
        }

        return response;
    }

    public Response deletePatients(String id) {
        System.out.println("...invoking deletePatients, Patient Id is... " + id);
        Long patientId = Long.parseLong(id);
        Patient patient = patients.get(patientId);
        Response response = null;
        if (patient != null) {
            patients.remove(patientId);
            response = Response.ok().build();
        } else {
            response = Response.notModified().build();
        }

        return response;
```

```
    }

    public Prescription getPrescription(String prescriptionId) {
        long id = Long.parseLong(prescriptionId);
        Prescription prescription = prescriptions.get(id);
        return prescription;
    }

}
```



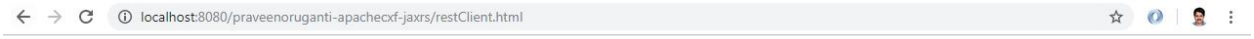
Welcome to JAX-RS Patient Services - Click Here For Available Services



Available SOAP services:

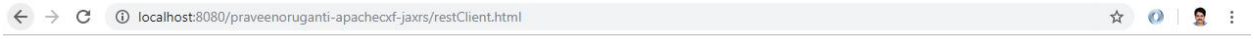
Available RESTful services:

Endpoint address: http://localhost:8080/praveenoruganti-apachecxf-jaxrs/services/patientservice
WADL : http://localhost:8080/praveenoruganti-apachecxf-jaxrs/services/patientservice?_wadl



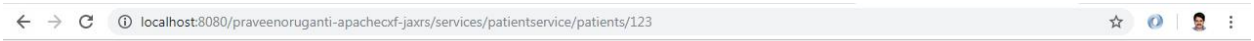
Patient Details:

Patient Id:
Patient Name:
read add delete



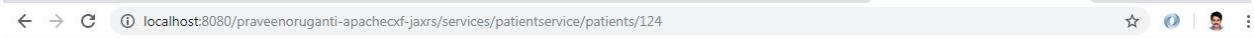
Patient Details:

Patient Id: 124
Patient Name: Praveen
read add delete



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Patient>
   <id>123</id>
   <name>John</name>
 </Patient>
```



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Patient>
   <id>124</id>
   <name>Praveen</name>
 </Patient>
```

You can view this code in my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-apachecxf-jaxrs-master)

Now let's see how to implement restful webservice using Jersey

Jersey is JAX-RS complaint and it is an open source from ORACLE.
Jersey at server side provides the servlet implementation which scans through predefined classes to identify restful resources.

We need to configure the jersey servlet in web,xml

praveenoruganti-jersey-jaxrs-master/pom.xml

```
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>

                <plugin>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.1</version>
                    <configuration>
                        <webXml>src/main/WebContent/WEB-INF/web.xml</webXml>
                    </configuration>
                </plugin>
            </plugins>
            <finalName>praveenoruganti-jersey-jaxrs</finalName>
        </build>
        <dependencyManagement>
            <dependencies>
                <dependency>
                    <groupId>org.glassfish.jersey</groupId>
                    <artifactId>jersey-bom</artifactId>
                    <version>2.18</version>
                    <type>pom</type>
                    <scope>import</scope>
                </dependency>
            </dependencies>
        </dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.glassfish.jersey.containers</groupId>
                <artifactId>jersey-container-servlet-core</artifactId>
            </dependency>
        </dependencies>
    </project>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This web.xml file is not required when using Servlet 3.0 container,
    see implementation details http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->
<web-app version="2.5" 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_2_5.xsd">
    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.praveen.jersey.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/restapi/*</url-pattern>
    </servlet-mapping>
</web-app>
```

index.jsp

```
<html>
<body>
    <h2>Hello Jersey RESTful Web Application!</h2>
    <p><a href="restapi/myresource">My resource</a>
</body>
</html>
```

MyResource.java

```
package com.praveen.jersey.rest;

import javax.ws.rs.GET;

@Path("myresource")
public class MyResource {

    @GET
    public String hello() {
        return "Hello Jersey";
    }
}
```

Let's create a client class

JerseyRestClient.java

```
package com.praveen.soap.client;

import javax.ws.rs.client.Client;

public class JerseyRestClient {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        String result = client.target("http://localhost:8080/praveenoruganti-jersey-jaxrs-master/restapi/myresource")
                .request().get(String.class);
        System.out.println(result);
    }
}
```

Output

Hello Jersey

You can view this code in my repository (https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-jersey-jaxrs-master)

RESTEASY

Resteasy is JAX-RS complaint and it is an open source from JBOSS.
Resteasy at server side provides the servlet implementation which scans through predefined classes to identify restful resources.

We need to configure the Resteasy servlet in web,xml

```xml
<dependencies>
    <dependency>
        <groupId>org.jboss.resteasy</groupId>
        <artifactId>resteasy-jaxrs</artifactId>
        <version>3.0.4.Final</version>
    </dependency>
</dependencies>
```

web.xml

```xml
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>Restful Web Application</display-name>

    <!-- Auto scan rest service -->
    <context-param>
        <param-name>resteasy.scan</param-name>
        <param-value>true</param-value>
    </context-param>

    <context-param>
        <param-name>resteasy.servlet.mapping.prefix</param-name>
        <param-value>/restapi</param-value>
    </context-param>

    <listener>
        <listener-class>
            org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap</listener-class>
    </listener>


    <servlet>
        <servlet-name>resteasy-servlet</servlet-name>
        <servlet-class>
            org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>resteasy-servlet</servlet-name>
        <url-pattern>/restapi/*</url-pattern>
    </servlet-mapping>

</web-app>
```

index.jsp

```jsp
<html>
<body>
    <h2>Hello RESTEASY RESTful Web Application!</h2>
    <p><a href="restapi/myresource">My resource</a>
</body>
</html>
```

MyResource.java

```java
package com.praveen.resteasy.rest;

import javax.ws.rs.GET;

@Path("myresource")
public class MyResource {

    @GET
    public String hello() {
        return "Hello RESTEASY";
    }
}
```

Let's create a client to test

RestClient.java

```
1  package com.praveen.soap.client;
2
3  import javax.ws.rs.client.Client;
5
6  public class RestClient {
7
8      public static void main(String[] args) {
9          Client client = ClientBuilder.newClient();
10         String result = client.target("http://localhost:8080/praveenoruganti-resteasy-jaxrs-master/restapi/myresource")
11                 .request().get(String.class);
12         System.out.println(result);
13     }
14 }
```

Output

Hello RESTEASY

You can view this code in my repository
(https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-resteasy-jaxrs-master)

Now let's develop the Hospital Management System restful webservice using Spring REST

Please refer the code present in my git repository
(https://github.com/praveenorugantitech/praveenorugantitech-webservice/tree/master/praveenoruganti-hms-restfulws-master)

Please note complete code for the above sections is present in my git repository
(https://github.com/praveenorugantitech/praveenorugantitech-webservice)

You can checkout my other ebooks in
https://praveenorugantitech.github.io/praveenorugantitech-ebooks/