



GROUND UP SERIES

Java™ Web Services

Syed Awase Khirni

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH

@sak008 | sak@sycliq.com/sak@territorialprescience.com | +91. 9035433124

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project (www.geo-spirit.org). He currently provides consulting services through his startup www.territorialprescience.com and www.sycliq.com. He empowers the ecosystem by sharing his technical skills worldwide, since 2008.



Terms of Use

- You shall not circulate these slides without written permission from Territorial Prescience Research I Pvt Ltd.
- If you use any material, graphics or code or notes from these slides, you shall seek written permission from TPRI and acknowledge the author Dr. Syed Awase Khirni
- If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material. If any of the material, that has been shared is further used for any unauthorized training by the recipient, he shall be liable to be prosecuted for the damages. Any supporting material that has been provided by the author, shall not be used directly or indirectly without permission.
- If this material, has been shared to any organization prior to the training and the organization does not award the contract to TPRI, it should not use the training material internally. If by any chance, the organization is using this training material without written permission, the organization is liable to pay for the damages to TPRI and is subjected to legal action, jurisdiction being Bangalore.
- Without unauthorized usage, TPRI has right to claim damages ranging from USD 50000 to USD 10,0000 dollars as damages.
- Any organization, which does not intend to go ahead with training or does not agree with the terms and conditions, should destroy the material from its network immediately. The burden of proof lies on the client, with whom this material has been shared.
- Recovery of the damages and legal fees will be born by the client organization/candidate/party, which has violated the terms and conditions.
- Only candidates who have attended the training session in person from Dr. Syed Awase Khirni, TPRI are entitled to hold this training material. They cannot further circulate it, or use it or morph it, or change it to provide trainings.
- TPRI reserves all the rights to this material and code plays and right to modify them as and when it deems fit.
- If you agree with the terms and conditions, please go ahead with using the training material. Else please close and destroy the slide and inform TPRI immediately.



Slide Version Updates

Last Updated	Version	Release Date	Updated by	Code Plays Done @
June 2016	1.7			
Sep 2017	1.8			



Program Agenda

Pre-requisites

- Core JAVA
- XML
- HTML

Workshop format

Configurations
25%

Workflow
30%

Programming
45%





SYED AWASE

I: JAVA WEB SERVICES: SET UP

Web Services Ecosystem



<http://axis.apache.org/axis/>



<http://projects.spring.io/spring-ws/>

<http://rest-assured.io/>



<http://cxf.apache.org>



<https://jersey.github.io>



Apache HttpClient

Apache HttpComponents Client

[https://mvnrepository.com/
artifact/org.apache.httpco
mponents/httpclient](https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient)



Software Requirements

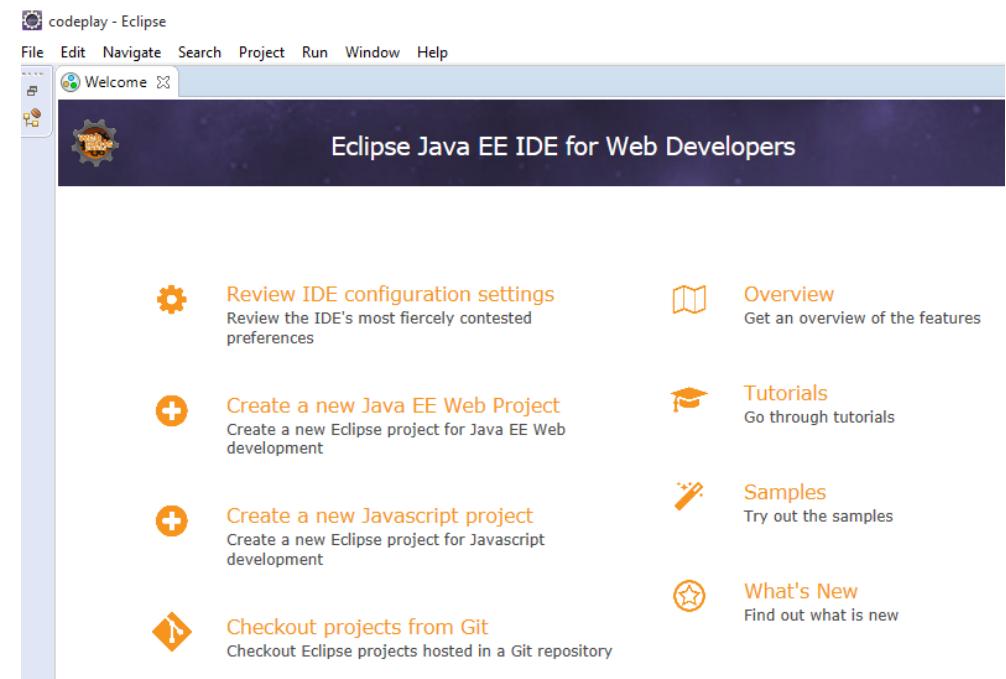
- JDK Latest 1.7/1.8
- Eclipse Oxygen
- Apache Tomcat
- Apache-CXF
- Apache Jersey
- JAX-WS
- JAX-RS
- Maven 3.2.5



Tools

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	jdk-8u144-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u144-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.65 MB	jdk-8u144-linux-i586.rpm
Linux x86	179.44 MB	jdk-8u144-linux-i586.tar.gz
Linux x64	162.1 MB	jdk-8u144-linux-x64.rpm
Linux x64	176.92 MB	jdk-8u144-linux-x64.tar.gz
Mac OS X	226.6 MB	jdk-8u144-macosx-x64.dmg
Solaris SPARC 64-bit	139.87 MB	jdk-8u144-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.18 MB	jdk-8u144-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u144-solaris-x64.tar.Z
Solaris x64	96.99 MB	jdk-8u144-solaris-x64.tar.gz
Windows x86	190.94 MB	jdk-8u144-windows-i586.exe
Windows x64	197.78 MB	jdk-8u144-windows-x64.exe

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>



Tools

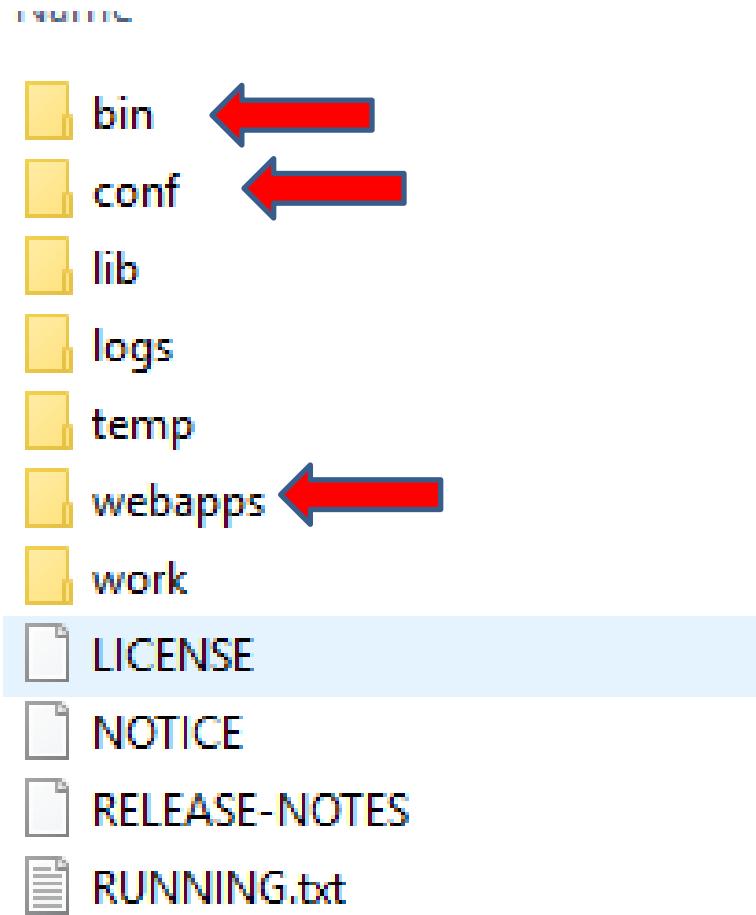
9.0.0.M26

Please see the [README](#) file for packaging information. It explains what ever

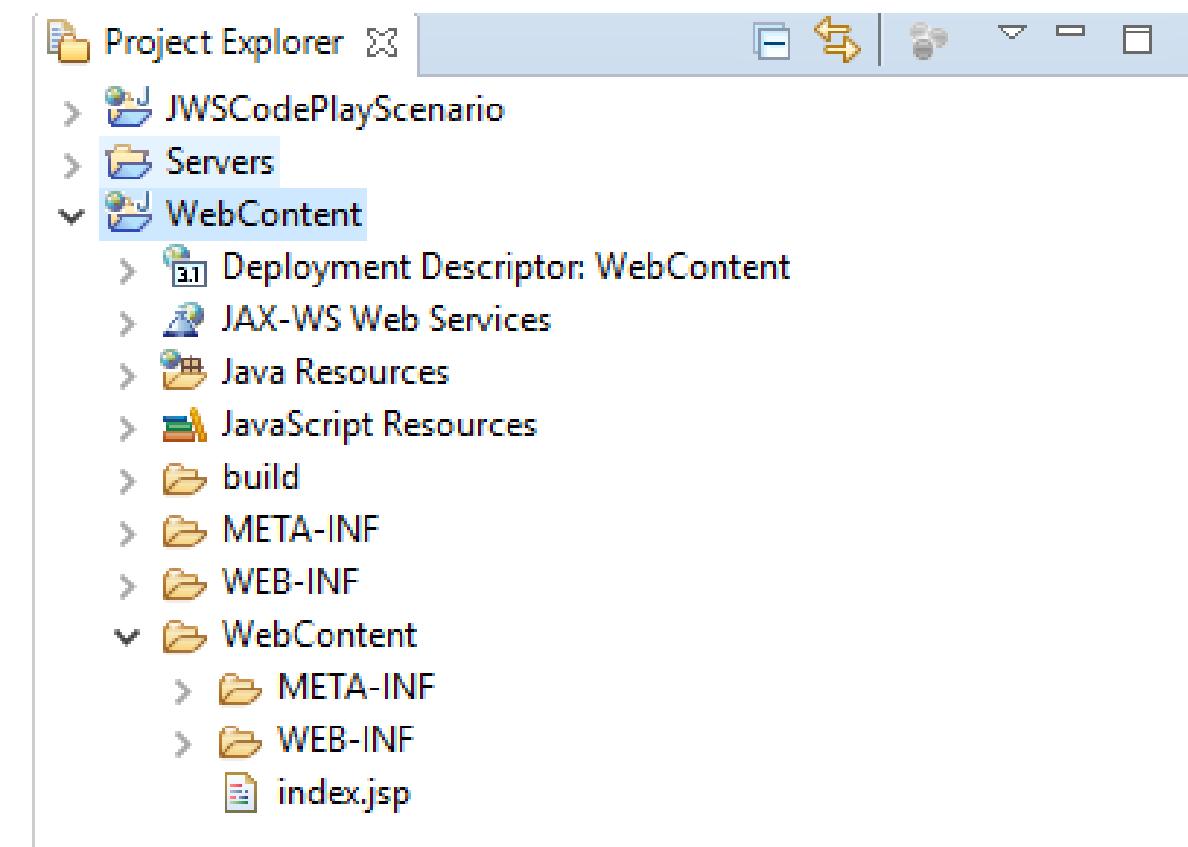
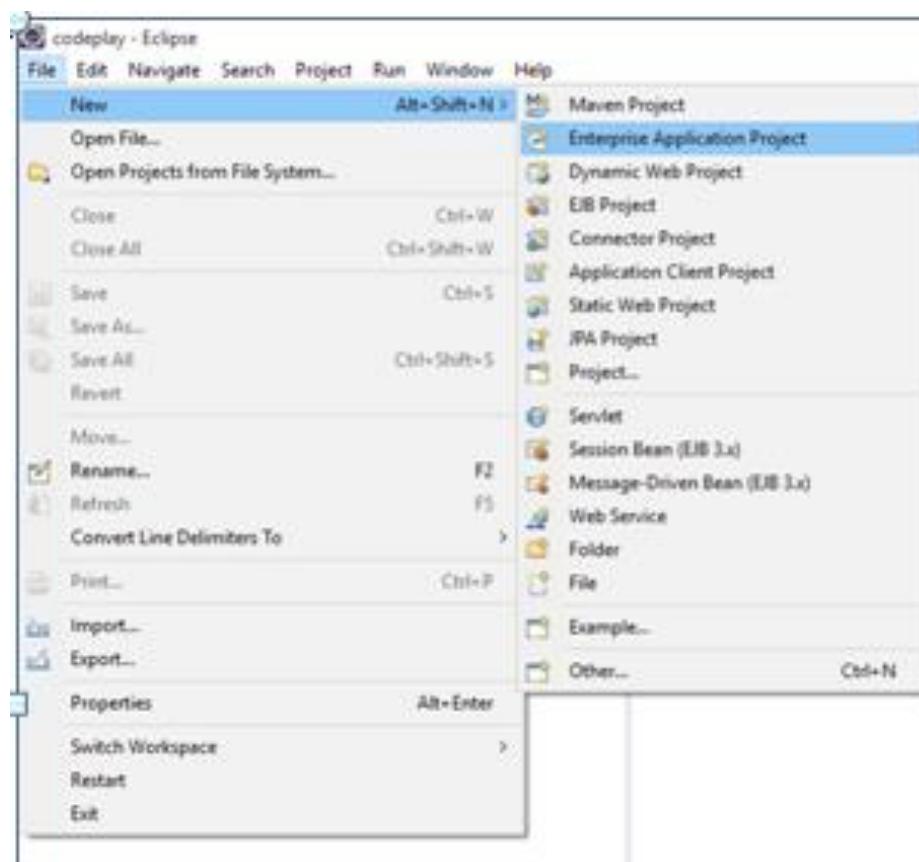
Binary Distributions

- Core:
 - [zip \(pgp, md5, sha1\)](#)
 - [tar.gz \(pgp, md5, sha1\)](#)
 - [32-bit Windows zip \(pgp, md5, sha1\)](#)
 - [64-bit Windows zip \(pgp, md5, sha1\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, md5, sha1\)](#)
- Full documentation:
 - [tar.gz \(pgp, md5, sha1\)](#)
- Deployer:
 - [zip \(pgp, md5, sha1\)](#)
 - [tar.gz \(pgp, md5, sha1\)](#)
- Extras:
 - [JMX Remote jar \(pgp, md5, sha1\)](#)
 - [Web services jar \(pgp, md5, sha1\)](#)
- Embedded:
 - [tar.gz \(pgp, md5, sha1\)](#)
 - [zip \(pgp, md5, sha1\)](#)

<http://tomcat.apache.org/download-90.cgi>



Enterprise Application Project



codeplay - WebContent/WebContent/index.jsp - Eclipse

File Edit Navigate Search Project Run Window Help

Project Explorer X Properties for WebContent

Project Facets Configuration: <custom>

Save As... Delete

Project Facet	Version
Axis2 Web Services	1.0
CXF 2.x Web Services	
Dynamic Web Module	3.1
EJBDoclet (XDoclet)	1.2.3
Java	1.7
JavaScript	1.0
JavaServer Faces	2.2
JAX-RS (REST Web Services)	1.1
JAXB	2.2
JPA	2.1
Utility Module	
Web Fragment Module	3.0
WebDoclet (XDoclet)	1.2.3

Details Runtimes

Apache Tomcat v9.0

Show all runtimes Make Primary New...

Runtime composition:
<no runtime selected>

Revert Apply

Apply and Close Cancel

Project Explorer X Properties for WebContent

Resource Builders Coverage Deployment Assembly Java Build Path Java Code Style Java Compiler Java Editor Javadoc Location JavaScript JSP Fragment Project Facets Project References Run/Debug Settings Server Service Policies Targeted Runtimes Task Repository Task Tags Validation Web Content Settings Web Page Editor Web Project Settings WikiText XDoclet



SOAP UI

SoapUI 5.3.0

File Project Suite Case Step Tools Desktop Help

Empty SOAP REST Import Save All Forum Trial Preferences Proxy

Search Forum Online Help

SoapUI Starter Page

Do More API Testing with Less
Save Time, Hassle, and Complexity with SoapUI Pro

SMARTBEAR
Service Virtualization
Made Simple with ServiceV
An Easy-to-Use Solution for Everyone on the Team

ServiceV Pro Read eBook

Whether your QA team depends on an internal development team or a 3rd party API, ServiceV can help speed delivery by reducing bottlenecks and allowing you to develop and test in parallel.

In this eBook, we'll cover:

- The challenges of service virtualization
- Why mocking is a workaround, not a solution
- Virtualizing your desktop quickly and easily
- Sharing your virtual services with the team
- Automating your virtual services with Jenkins, TFS, Bamboo

READ NOW

Getting Started with SoapUI

Welcome to the world's most popular API testing tool! Whether you've got SOAP or REST based web services, we'll show you how SoapUI can help you quickly get started with:

- Creating your first project
- Functional testing
- Load testing

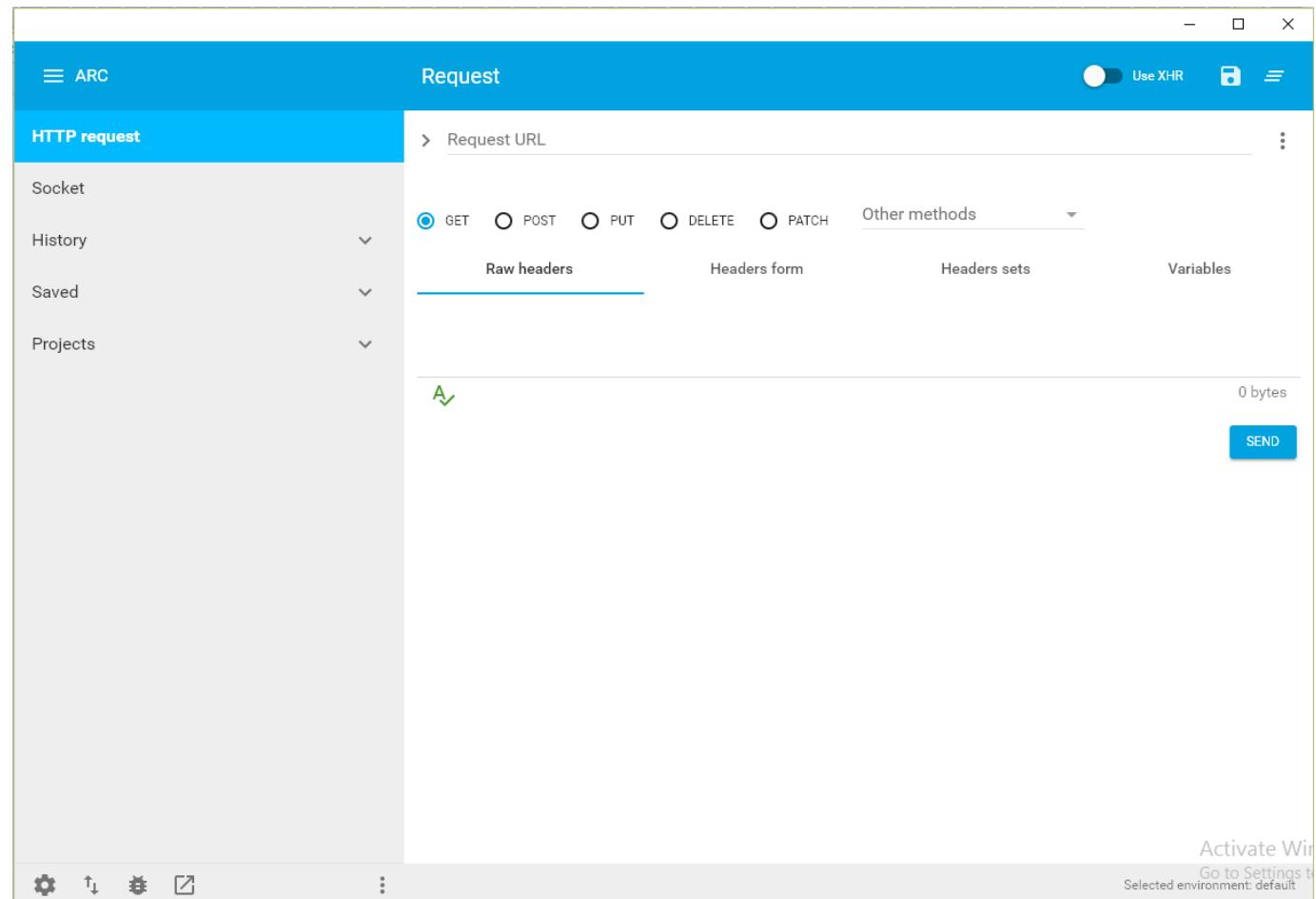
Activate Windows
Go to Settings to activate Windows.

Workspace Properties

Property	Value
Name	SycliQWorkspace
Description	
File	E:\CT\SOAP-UI\Code\
Project Root	



Advanced REST Client



POSTMAN

The screenshot shows the Postman application interface. The left sidebar displays a navigation menu with options like History, Collections, All, Me, Team, and various API collections such as 'GSMArena ...' (3 requests), 'brands', 'devices', 'specs', 'Customer' (1 request), 'gsmarea-collection' (0 requests), 'GSMArena API co...' (3 requests), and 'Postman Echo' (20 requests). The main workspace is titled 'Builder' and shows a collection named 'GSMArena ...'. Inside the collection, there is a single request titled 'Get devices from a brand'. The request details are as follows:

- Method: GET
- URL: https://gsmarena-obrisiswox.now.sh/specs/lenovo_zuk_edge-8423
- Authorization: No Auth
- Headers: (1)
- Body
- Pre-request Script
- Tests

The response area contains the placeholder text: "Hit the Send button to get a response." Below the response area are four buttons: Share, Mock, Monitor, and Document. At the bottom right of the workspace, there is a message: "Activate Windows Go to Settings to activate Windows." The top right corner of the window shows a user profile for "Syed Awase..." and a sync status indicator.





SYED AWASE

II: JAVA WEB SERVICES: INTRODUCTION

A Business process demands cross-domain collaboration

**Technology required to support
cross-channel/platform business logic**



Business Objective

- Provide flexibility in business processes towards an **Agile Organization**
- **Decrease in IT Costs**
- **Reusability and Scalability of the Applications**
- Customer-centric business applications that enhance user experience and retain customers in the long run.
- Effective communication mechanisms between different business departments to streamline operations.



Objectives

#1. BPM STRATEGY

- Aligning your software applications with business processes to achieve a specific objective
- Strategize the relevance of a business process and create a holistic architectural framework.

#2. AGILE ORGANIZATION

- Proactively investigate demands and opportunities and organize business process to adapt to them.
- Focus on interoperable, standardized, reusable, composable and scalable solutions.

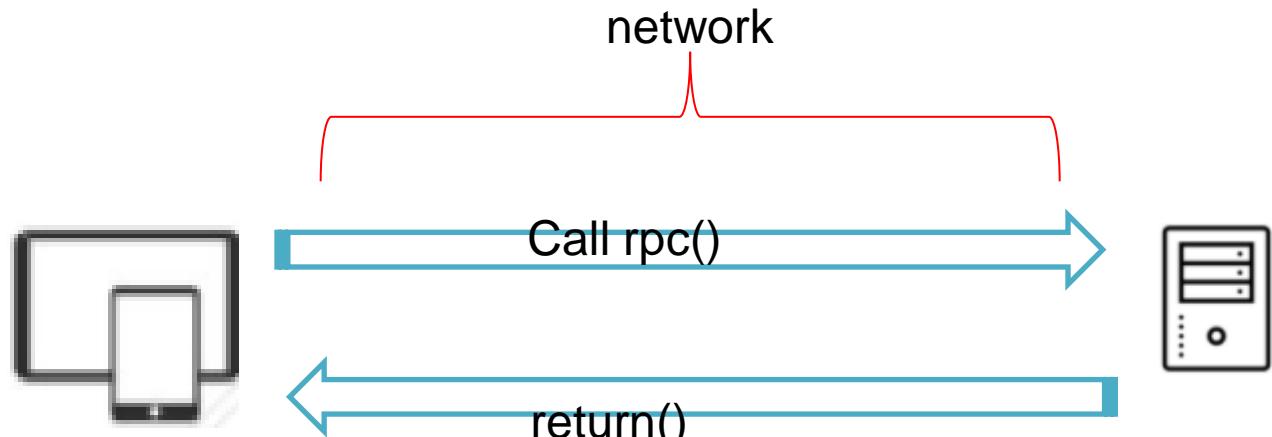
#3. COST-TIME TO MARKET

- Focus on reduction in developmental time and cost.
- interoperability, reusability, scalability, adaptive business process enable quick deployment of the product to the market.



RPC (REMOTE PROCEDURE CALLS)

- A powerful technique that provides distributed computing capabilities across a network of machines.
- It is a form of inter-process communication that enables function calls between applications that are located across different(or the same) locations over a network.



Language Barriers

XML is a standardized language that is used for message exchange.

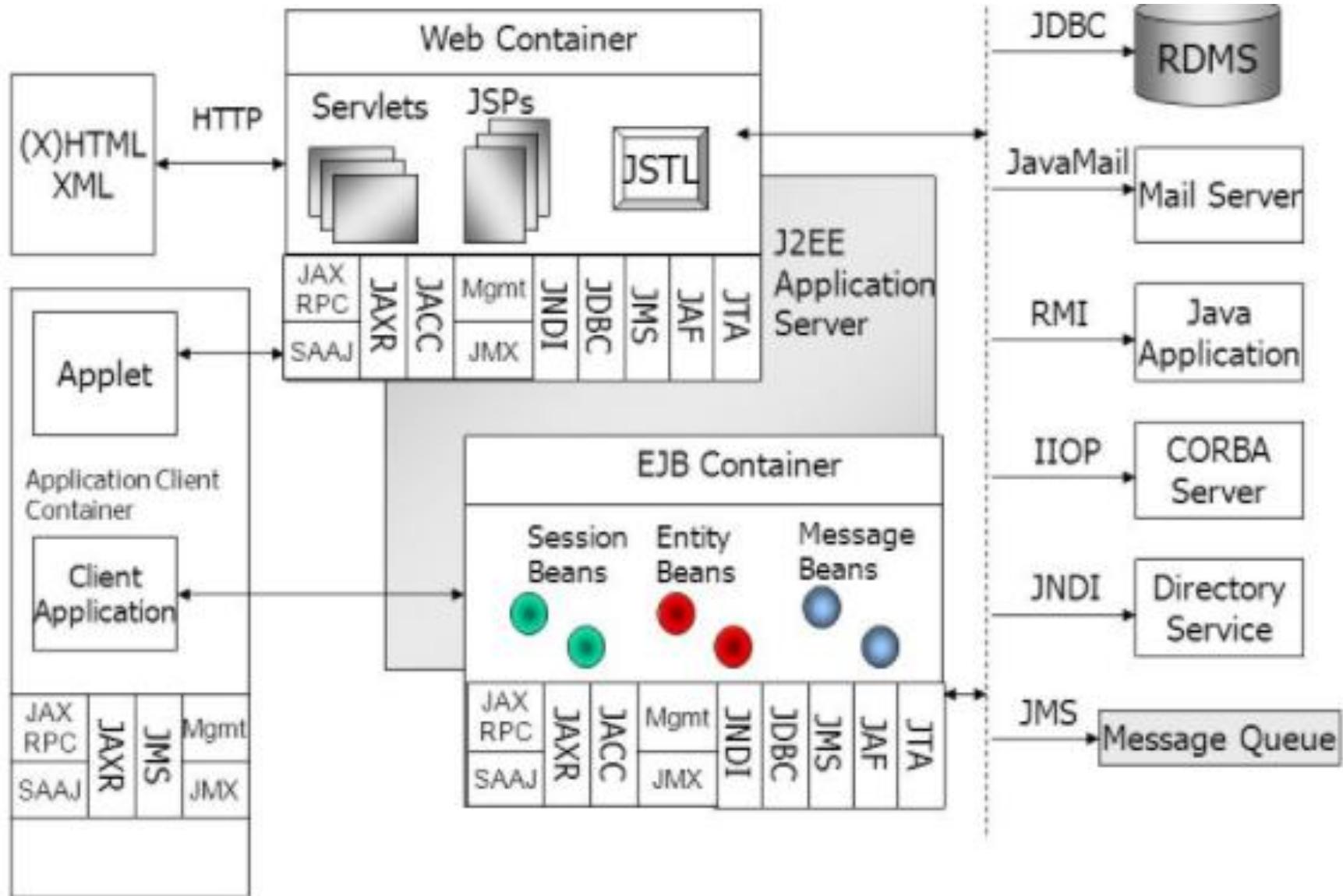
Platform Barrier

SOAP has been implemented on many platforms (UNIX, WINDOW, MAC)

Network Barrier

HTTP and SMTP are standardized network protocols





JAX-WS Evolution

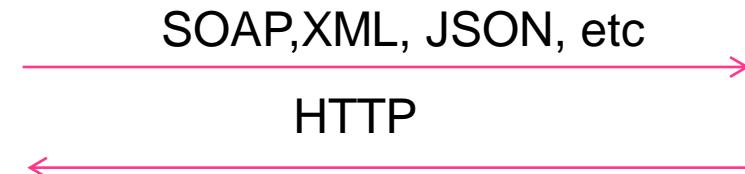
	Year				
JSR 101	2002	JAX-RPC	SOAP/HTTP-HTTPS	XML/XSD	WSDL WS-I-Basic Profile
JSR 224	2006	JAX-WS, JAXM, JAXB	SOAP1.2	XML/XSD	WSDL2.0
JSR 311	2008	JAX-RC			
JSR 339		JAX-RC 2.0			



Web Service

Original Series

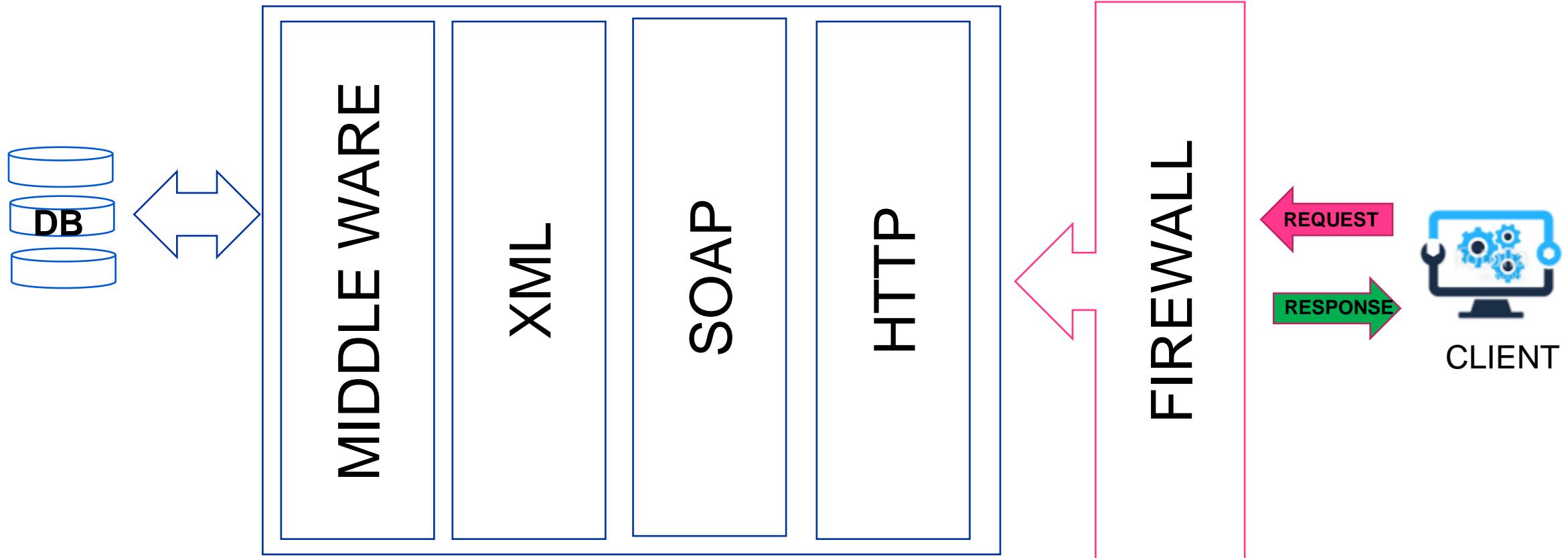
Consumer Application/
Client



Provider
Application
/Server



Web Service



Web Service

- They are self-contained, modular, distributed, dynamic applications that can be described, published, located or invoked over the network to create products, processes and supply chains.
- The application can be local, distributed or web-based.
- Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML and XML.
- They are XML based information exchange systems that use the internet for direct application-to-application interaction. These systems can include programs, objects, messages or documents.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.



STACK

WEB SERVICE TECHNOLOGY

DISCOVERY

DESCRIPTION

PACKAGING

TRANSPORT

NETWORK

TCP/IP NETWORK MODEL

APPLICATION

TRANSPORT

NETWORK

LINK



TECHNOLOGY

XML

- Structured data representation tied to a schema
- Works well with SOAP Protocol, a **de facto standard for implementing web service.**

XKMS

- XML Key Management Services are a set of security and trust related services that add Private Key Infrastructure (PKI) capabilities to web services.



TECHNOLOGY

SAML

- Security Assertions Markup Language is an XML grammar for expressing the occurrence of security events, ex: an authentication event.
- Used within the web services architecture, it provides a standard flexible authentication system.

XML-Dsig

- XML Digital Signatures allow any XML document to be digitally signed.



TECHNOLOGY

XML-Enc

- XML Encryption specification allows XML data to be encrypted and for the expression of encrypted data as XML.

XSD

- XML Schemas are an application of XML used to express the structure of XML documents.



TECHNOLOGY

P3P

- W3C platform for Privacy Preferences is an XML grammar for the expression of data privacy policies.

WSDL

- An XML format for describing network services as a set of end points operating on messages containing either document-oriented or procedure-oriented information.



TECHNOLOGY

JABBER

- A new lightweight, asynchronous transport protocol used in peer-to-peer applications.

WSFL

- Web Services Flow Language is an extension to WSDL that allows for the expression of work flows within the web services architecture.



TECHNOLOGY

ebXML

- A suite of XML-based specifications for conducting electronic business. Built to use SOAP, ebXML offers one approach to implementing business-to-business integration services.

UDDI

- Universal Description, Discovery and Integration Services.



TECHNOLOGY

WADL

- A machine readable XML description of HTTP-based web services. It models the resources provided by a services and the relationships between them.
- It is intended to simply the reuse of web services that are based on existing HTTP architecture of the web.
- **REST** equivalent of SOAP's web services description language.

ENDPOINT

- A web service endpoint is a web address (URL) at which clients of a specific service can gain access to it.
- By referencing the URL, clients can get to operations provided by that service.
 - Port: A Unique endpoint with its own address.
 - Protocol: is a specific way to interact
 - Message: a piece of abstract lingo that aids communication
 - Port Type: what type of operations a post can perform
 - Operation: an action a service can perform
 - Service: a group of endpoints that share something in common



Web Services Components

SOAP

- Simple Object Access Protocol

UDDI

- Universal Description, Discovery and Integration

WSDL

- Web Services Description Language



XML to tag data

SOAP to transfer a message

WSDL to describe the availability of service



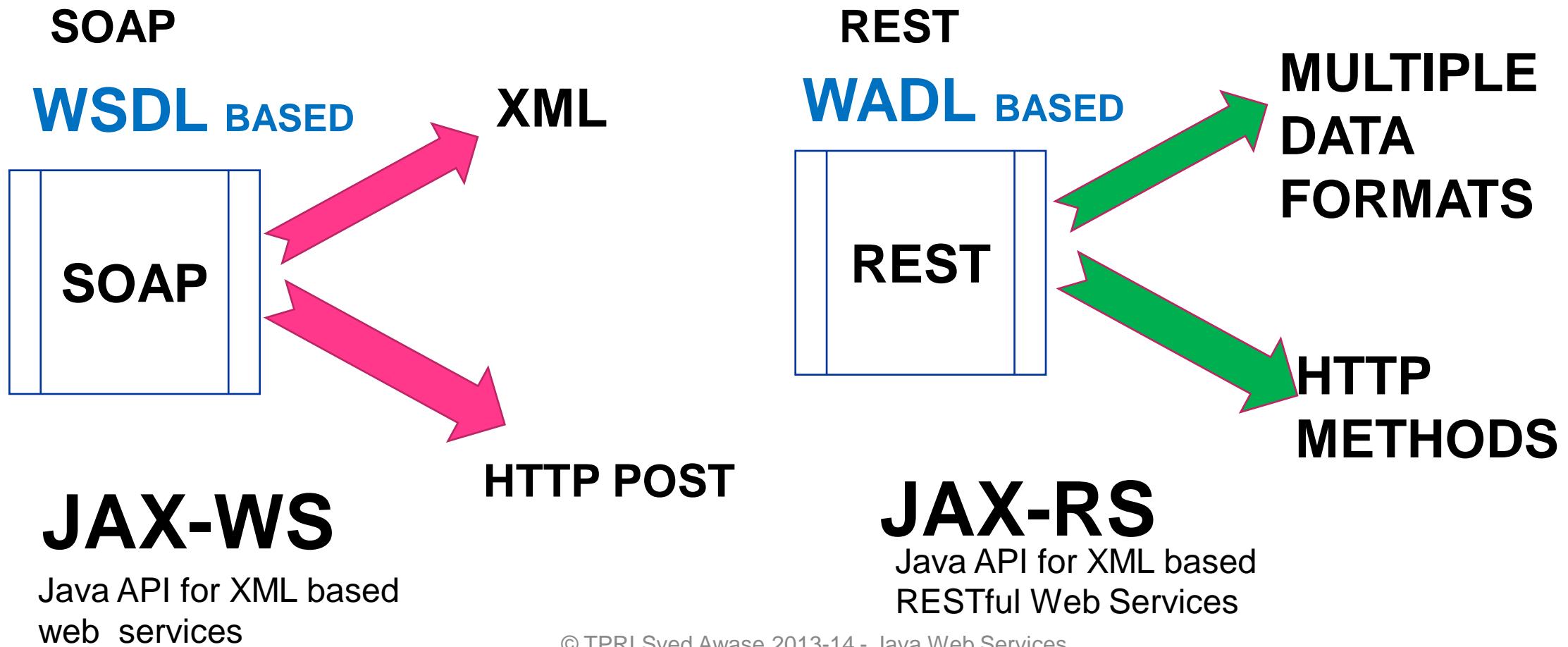
Why Web Services?

- 1 • Interoperability
- 2 • Standardized Protocol
- 3 • Low cost Communication
- 4 • Loosely coupled
- 5 • Coarse-grained
- 6 • Asynchronous/Synchronous Configuration
- 7 • Support Remote Procedure Calls
- 8 • Supports Document Exchange

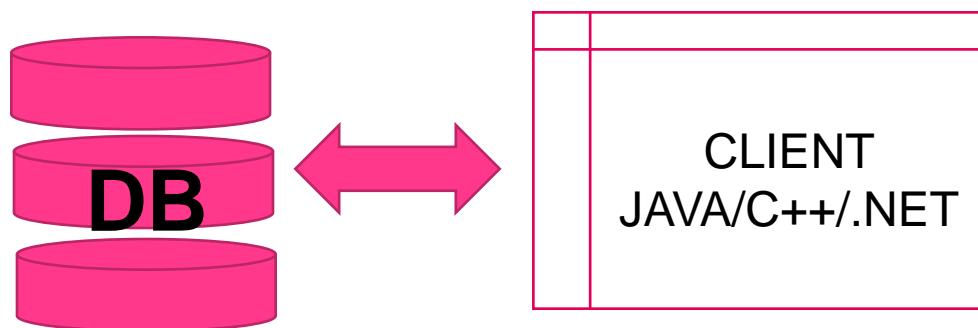


Web Services: approaches

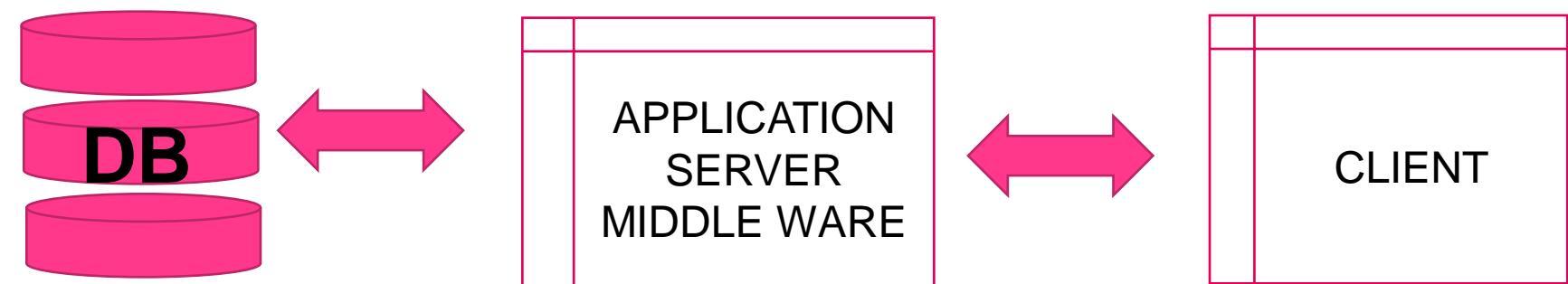
Original Series



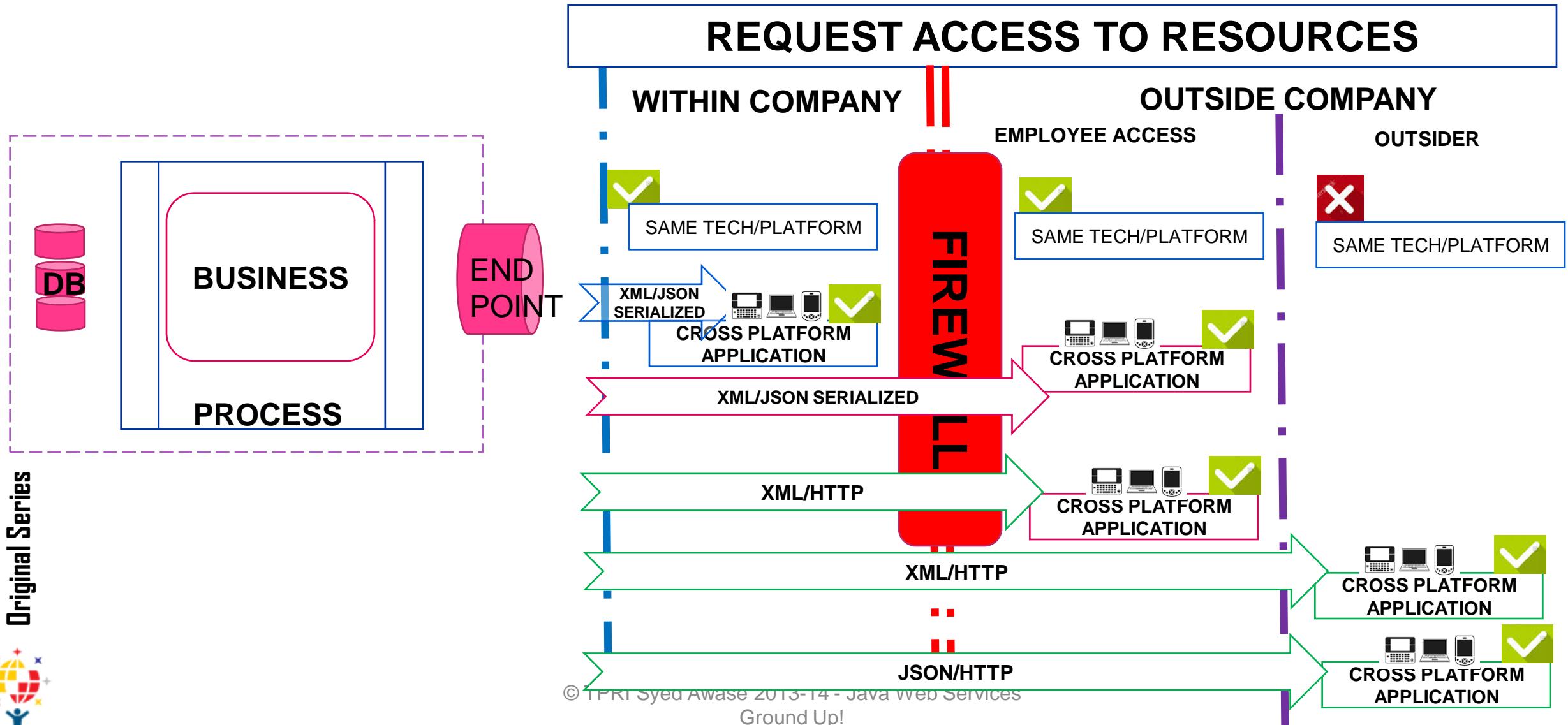
2- Tier Architecture



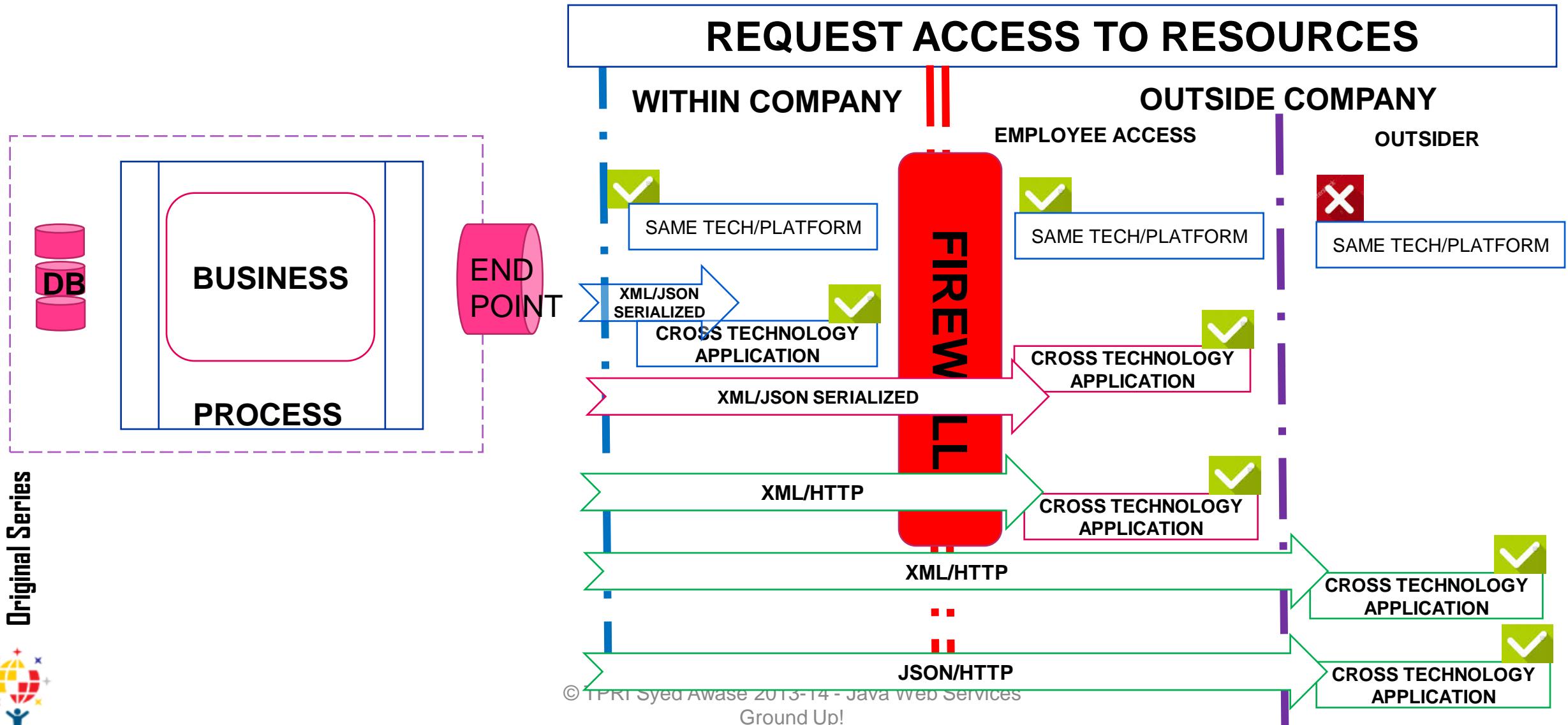
3- Tier Architecture



Need for Web Services



Need for Web Services

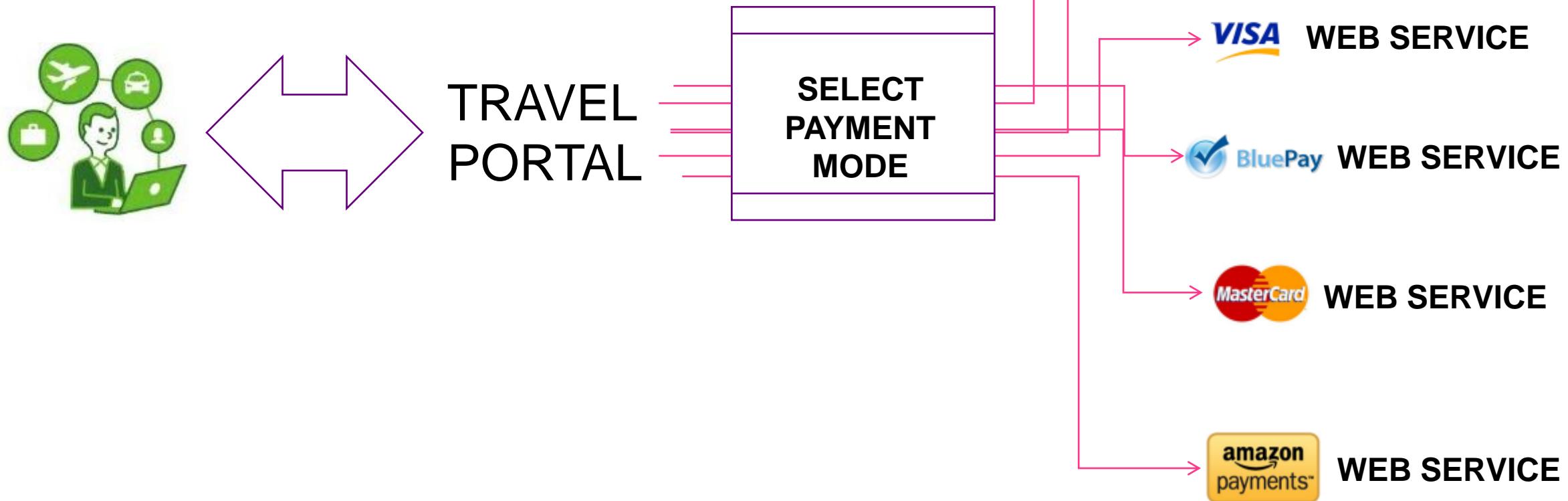


WHEN TO USE WEB SERVICES

- interoperability of applications over internet /intranet
- progressive web application development
- loosely coupled application architectures
- for communication between different platforms owned by different organization/within organization
- where scalability, security need to be ensured.



A Typical Scenario



XML-RPC

- XML-based protocol for exchanging information between computers.
- A simple protocol that uses XML messages to perform remote procedure calls
- Requests are encoded in XML and sent via HTTP POST
- They are embedded in the body of the HTTP response.
- XML-RPC is platform-independent and allows diverse applications to communicate
- Easiest way to communicate and get started with web services.



SOAP

- SOAP stands for Simple Object Access Protocol
- An XML based protocol for exchanging information between computers.
- SOAP is a communication protocol for communication between application.
- It is a format for sending messages and is designed to communicate via HTTP
- SOAP is platform, language independent and is simple and easily extensible.
- SOAP allows you to get around firewalls.



UDDI

- UDDI stands for Universal Description, Discovery and Integration is an XML-based standard for describing, publishing and finding web services.
- It is a platform independent, open framework for defining a distributed registry of web services.
- UDDI can communicate via SOAP, CORBA and Java Remote Method Invocation Protocol.
- It uses WSDL to describe interfaces to web services. It is one of the three foundation standards for web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the internet.



WSDL

- WSDL stands for Web Services Description Language and was developed jointly by Microsoft and IBM.
- An XML-based language for describing web services and how to access them.
- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is a standard format for describing a web service, how to access a web services and what operations it will perform.
- A Language for describing how to interface with XML-based services. It is an integral part of UDDI, an XML-based worldwide business registry.
- UDDI uses WSDL for describing web services and how to access them.



SYED AWASE

II: XML : EXTENSIBLE MARKUP LANGUAGE

XML USAGE

XML

XSD

NAMESPACES

Data Manipulation

CONFIGURATION

DATA EXCHANGE

Saving Structured
Data

- Custom Markup Using XML

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```



XML SCHEMA DEFINITION (XSD)

- XSD provides validity and vocabulary to an XML document.
- XSD can
 - be extensible for future additions
 - be richer and more powerful than DTD
- XSD supports data types
- XSD is written in XML
- XSD supports namespace and is a W3C recommendation
- XSD Defines
 - Elements
 - Attributes
 - Namespaces
 - Order
 - Number of Occurrences
 - Restrictions



XSD Example

```
<?xml version = "1.0"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
    targetNamespace = "http://www.syycliq.com"
    xmlns = "http://www.syycliq.com" elementFormDefault = "qualified">
        <xs:element name = 'class'>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name = 'student' type = 'StudentType' minOccurs = '0'
                        maxOccurs = 'unbounded' />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:complexType name = "StudentType">
            <xs:sequence>
                <xs:element name = "firstname" type = "xs:string"/>
                <xs:element name = "lastname" type = "xs:string"/>
                <xs:element name = "nickname" type = "xs:string"/>
                <xs:element name = "marks" type = "xs:positiveInteger"/>
            </xs:sequence>
            <xs:attribute name = 'rollno' type = 'xs:positiveInteger' />
        </xs:complexType>
    </xs:schema>
```



XML Document based on XSD

```
<?xml version = "1.0"?>
<class xmlns = "http://www.sycliq.com"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation = "http://www.sycliq.com student.xsd">
    <student rollno = "1">
        <firstname>Awase</firstname>
        <lastname>Syed</lastname>
        <nickname>aicy</nickname>
        <marks>93</marks>
    </student>
    <student rollno = "2">
        <firstname>Ameese</firstname>
        <lastname>Syed</lastname>
        <nickname>munna</nickname>
        <marks>95</marks>
    </student>
    <student rollno = "3">
        <firstname>Azeez</firstname>
        <lastname>Syed</lastname>
        <nickname>abu</nickname>
        <marks>90</marks>
    </student>
</class>
```



Type Definitions in XSD

- Element : Simple Element can contain only text. It cannot contain any other element
- Attribute : It is a type in itself and is used in complex element
- Restriction: it defines the acceptable values of an XML element.
- Complex Element : It is an XML element which can contain other elements and/or attributes. A complex element can be created by
 - Defining a complex type and then create an element using type attribute
 - Defining a complex type by naming directly



XSD: Complex Type

Complex Type

```
<xs:complexType name = "SensorType">
  <xs:sequence>
    <xs:element name = "sensorid" type = "xs:string"/>
    <xs:element name = "sensorname" type = "xs:string"/>
    <xs:element name = "location" type = "xs:string"/>
    <xs:element name = "reading" type = "xs:Integer"/>
  </xs:sequence>
  <xs:attribute name = 'sensorid' type = 'xs:positiveInteger' />
</xs:complexType>

<xs:element name = 'sensor' type = 'SensorType' />
```

Complex Name

```
<xs:element name = "sensor">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "sensorid" type = "xs:string"/>
      <xs:element name = "sensorname" type = "xs:string"/>
      <xs:element name = "location" type = "xs:string"/>
      <xs:element name = "reading" type = "xs:Integer"/>
    </xs:sequence>
    <xs:attribute name = 'sensorid' type = 'xs:positiveInteger' />
  </xs:complexType>
<xs:element>
```



XSD: Complex Types

	Type	Description
1	Empty	Complex Empty complex type element can only have attributes but no contents
2.	Elements Only	Elements-Only complex type element can only contain elements
3.	Text only	Text-Only complex type element can only contain attribute and text
4	Mixed	Mixed complex type element can contain element, attribute, text
5	Indicators	Indicators controls the ways how elements are to be organized in an XML document
6	<any>	The <any> element is used for elements which are not defined by schema
7	<anyAttribute>	The <anyAttribute> attribute is used for attribute which are not defined by schema





XSD: String <xs:string>

- It can take characters, line feeds, carriage returns, tab characters.
- Restrictions
 - Enumerations
 - Length
 - maxLength
 - minLength
 - Pattern
 - whiteSpace

ID	Represents the ID attribute in XML and is used in schema attributes
IDREF	Represents the IDREF attribute in XML and is used in schema attributes
language	Represents a valid language id
Name	Represents a valid XML name
NMTOKEN	Represents a NMTOKEN attribute in XML and is used in schema attributes
normalizeString	Represents a string that does not contain line feeds, carriage returns, or tabs
String	Represents a string that can contain line feeds, carriage returns, or tab
Token	Represents a string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces or multiple spaces.



XSD:Date Time <xs:date>

- It is used to represent data in YYYY-MM-DD format
- <xs:time>
 - Is used to represent time in hh:mm:ss format
- Restrictions
 - Enumeration
 - maxExclusive
 - minInclusive
 - minExclusive
 - Pattern
 - whiteSpace
- <xs:datetime>
 - Is used to represent date and time in YYYY-MM-DDThh:mm:ss format
- <xs:duration>
 - Is used to represent time interval in PnYnMnDTnHnMnS format

```
<startTime>1980-03-23T10:20:15</startTime>
```

```
<period>P3Y6M4DT9H10M30S</period>
```

XSD:Numeric Data Types

byte	Signed 8 bit integer
Decimal	Decimal value
Int	Signed 32 bit integer
integer	An integer value
Long	A signed 64 bit integer
negativeInteger	An integer having only negative values
nonNegativeInteger	An integer having only non-negative value
nonPositiveInteger	An integer having only non-positive values
positiveInteger	An integer having only positive values

Short	A signed 16 bit integer
unsignedLong	An unsigned 64 bit integer
unsignedInt	An unsigned 32 bit integer
unsignedShort	An unsigned 16 bit integer
unsignedByte	An unsigned 8 bit integer



XSD

```
<xsd:element name="User_dob" type="xsd:date"/>
```

```
<xsd:element name="User_address" type="xsd:string"/>
```

```
<xsd:element name="OrderID" type="xsd:int"/>
```

```
<User_dob>2000-01-12T12:13:14Z</User_dob>
```

```
<User_address> #8 Cessna Business Park, Bangalore</User_address>
```

```
<OrderID>  
12314124312312  

```

```
<xsd:element name="Description" type="xsd:string"/>
```

```
<Body>  
| Work Harder, Party Hard!!  
</Body>
```



XSD

```
<xsd:element name="Customer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="CustomerName" type="xs:string" />
      <xsd:element name="Dob" type="xs:date" />
      <xsd:element name="Address" type="xs:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Supplier">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="SupplierName" type="xs:string" />
      <xsd:element name="Phone" type="xs:integer" />
      <xsd:element name="Address" type="xs:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML

```
<Customer>
  <CustomerName>Syed Awase</CustomerName>
  <Dob>11/11/1980</Dob>
  <Address>Umiya Business Bay, Cessna
Business Park</Address>
</Customer>

<Supplier>
  <SupplierName>SYCLIQ GEOSPATIAL PVT LTD</
SupplierName>
  <Phone>9035433124</Phone>
  <Address>CESSNA BUSINESS PARK</Address>
</Supplier>
```



XSD

- **Cardinality:** specifying how many times an element can appear

```
<xs:element name="User_dob"
    type="xs:date"/>

<xs:element name="Todo_Tasks"
    type="xs:string"
    minOccurs="2"
    maxOccurs="10"/>

<xs:element name="Basket_Items"
    type="xs:integer"
    minOccurs ="0"
    maxOccurs="unbounded"/>
```

XML



XSD

- Mixed Element Content

```
<xs:element name="MarkedUpDesc">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Bold" type="xs:string" />
      <xs:element name="Italic" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML

```
<MarkedUpDesc>
This is an <Bold>Example</Bold> or
| <Italic>Mixed</Italic> Content,
Note there are elements mixed in with the elements data.
</MarkedUpDesc>
```



XSD

- Simple Type

```
<xs:simpleType name="LetterType">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z]" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="SizeByNumberType">
    <xs:restriction base="xs:positiveInteger">
        <xs:maxInclusive value="21"/>
    </xs:restriction>
</xs:simpleType>
```

XML

```
<xs:simpleType name="SizeByStringNameType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
        <xs:enumeration value="large"/>
    </xs:restriction>
</xs:simpleType>
```



XSD

XML

```
<xs:simpleType name="PaymentMethodType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="VISA" />
    <xs:enumeration value="MasterCard" />
    <xs:enumeration value="Cash" />
    <xs:enumeration value="Amex" />
  </xs:restriction>
</xs:simpleType>
```



XSD-Miscellaneous Data Types

<xs:boolean>	
base64Binary	Represents base 64 encoded binary data
hexBinary	Represents hexadecimal encoded binary data
<xs:anyURI>	Data type is used to represent URI



XSD:Demo1

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.syqliq.com/Customer"
         xmlns:tns="http://www.syqliq.com/Customer" elementFormDefault="qualified">
    <element name="customer" type="tns:Customer"></element>
    <complexType name="Customer">
        <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="address" type="string" />
            <element name="gender" type="string" />
            <element name="phone" type="string" />
            <element name="city" type="string" />
            <element name="country" type="string" />
        </sequence>
    </complexType>
</schema>
```



XML Document for XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:customer xmlns:tns="http://www.sycliq.com/Customer"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.sycliq.com/Customer Customer.xsd ">
    <tns:id>1</tns:id>
    <tns:name>Syed Awase Khirni</tns:name>
    <tns:age>39</tns:age>
    <tns:dob>2001-01-01</tns:dob>
    <tns:email>awasekhirni@gmail.com</tns:email>
    <tns:address>Bangalore</tns:address>
    <tns:gender>M</tns:gender>
    <tns:phone>+919035433124</tns:phone>
    <tns:city>Bangalore</tns:city>
    <tns:country>India</tns:country>
</tns:customer>
```



XSD:Demo2

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.syqliq.com/Patient"
         xmlns:tns="http://www.syqliq.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>

    <simpleType name="UID">
        <restriction base="int">
            <pattern value="[0-9]"></pattern>
        </restriction>
    </simpleType>

    <simpleType name="String25Chars">
        <restriction base="string">
            <maxLength value="25"/>
        </restriction>
    </simpleType>

    <simpleType name="Gender">
        <restriction base="string">
            <enumeration value="M"/>
            <enumeration value="F"/>
            <enumeration value="N"/>
        </restriction>
    </simpleType>
</schema>
```



XML for XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:patient xmlns:tns="http://www.syqliq.com/Patient"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.syqliq.com/Patient Patient.xsd ">
    <tns:id>1</tns:id>
    <tns:name>Murugan</tns:name>
    <tns:age>23</tns:age>
    <tns:dob>2001-01-01</tns:dob>
    <tns:email>test@gmail.com</tns:email>
    <tns:gender>M</tns:gender>
    <tns:phone>123114323</tns:phone>
</tns:patient>
```



SYED AWASE

III: SOA: SERVICE ORIENTED ARCHITECTURES



SERVICE ORIENTED COMPUTING(SOC)

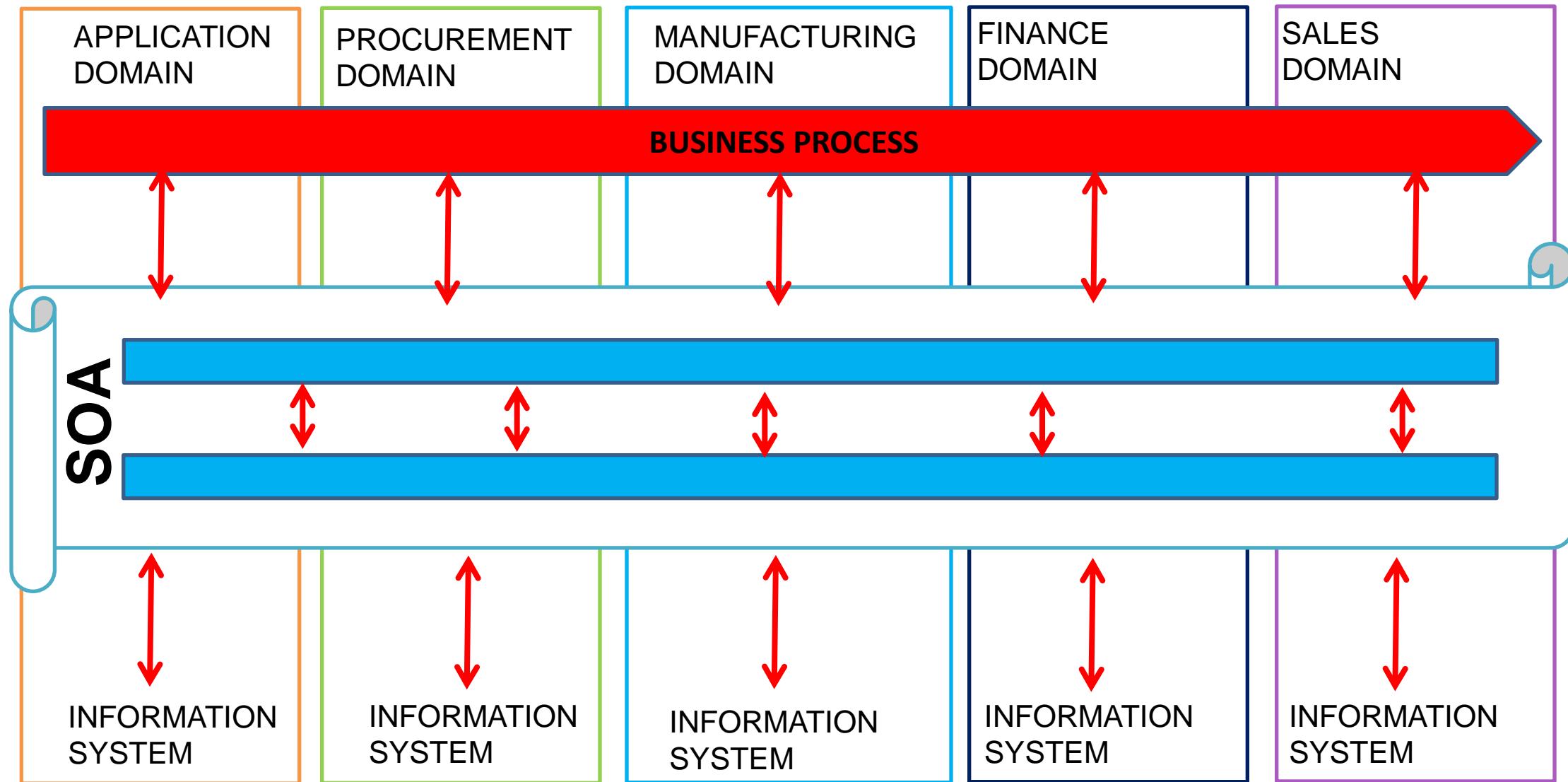
- SERVICE ORIENTED COMPUTING
- DISTRIBUTED COMPUTING
- TARGETS FLEXIBILITY/AGILITY

SERVICE ORIENTATION (SO)

- SERVICE ORIENTATION
- SET OF DESIGN PRINCIPLES
- HELPS ACHIEVE SOC GOALS

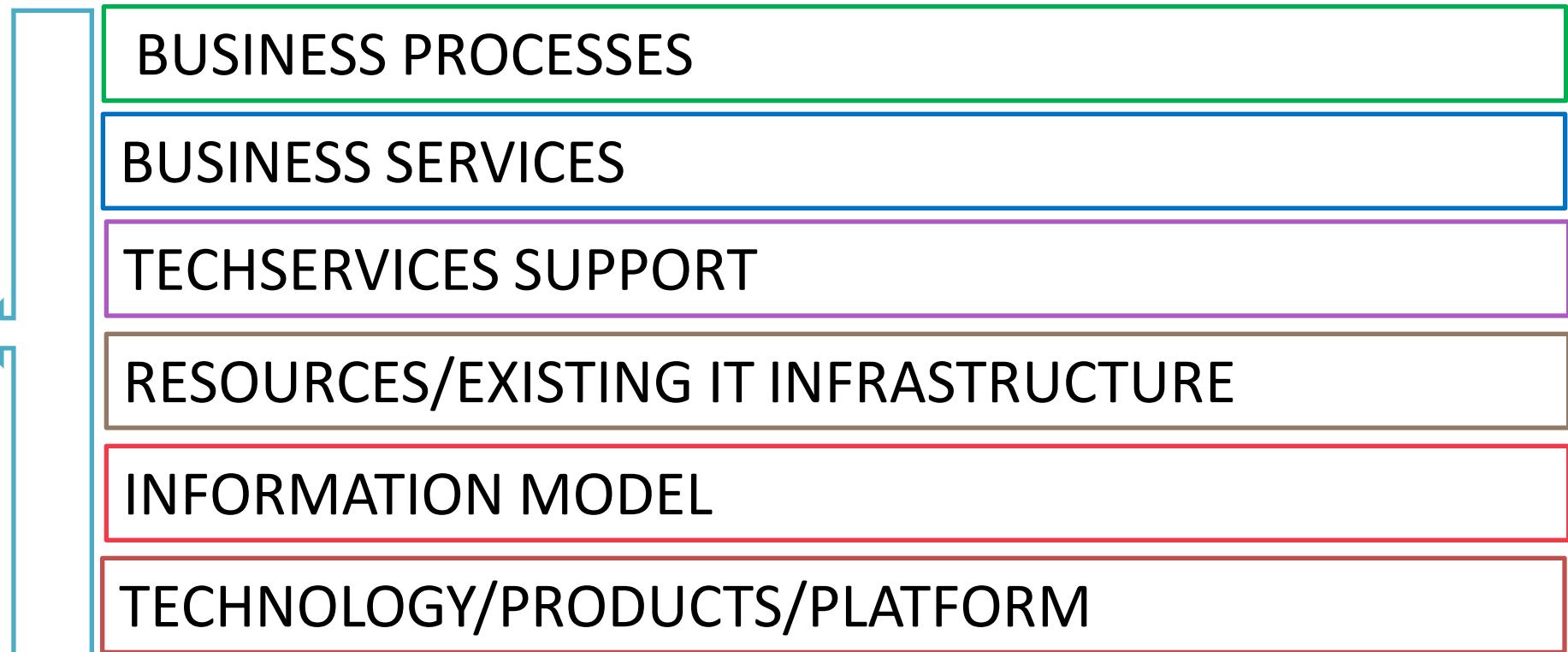
SERVICE ORIENTED ARCHITECTURE (SOA)

- SERVICE ORIENTED ARCHITECTURE
- ARCHITECTURE STYLE
- USES SO-BASED SERVICES
- STATIC/DYNAMIC ASPECTS
- IT IS TECHNOLOGY INDEPENDENT.
- EASILY REUSABLE, SCALABLE AND LOOSELY COUPLED

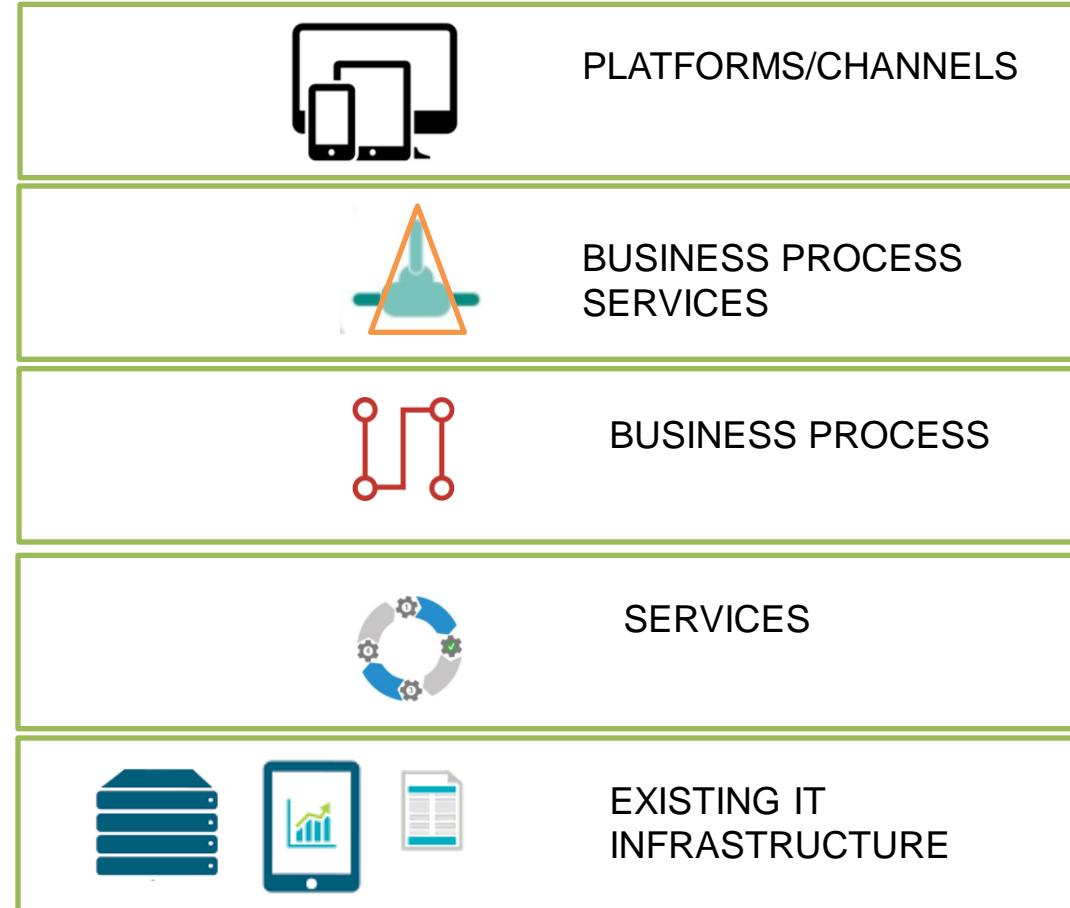


SOA BUILDING BLOCKS

SOA



SOA REFERENCE ARCHITECTURE



LOOSE COUPLING



CHARACTERISTICS OF PROCESS SERVICES

- Helps in encapsulating entire business process efficiently
- Task automation or robotic process automation of the business process typically implemented via an Orchestration engine
- Rich in composition and customization to adapt to the market demands
- Asynchronous call backs
- Scalable and reusable



Granularity?

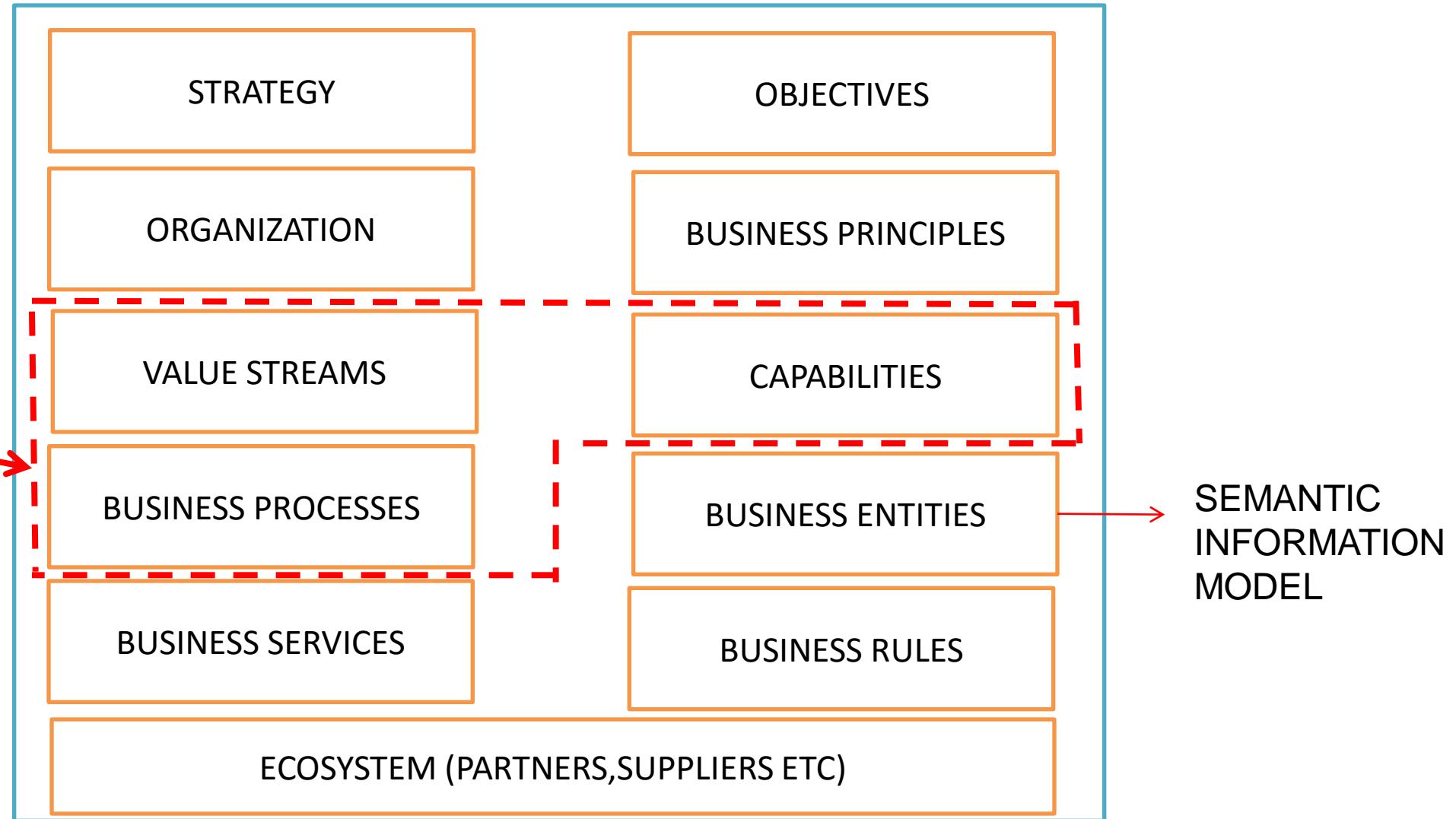
- Functional Granularity
 - Scope of functionality
 - Coarse-grained
 - Fine-grained
- Data Granularity
 - Amount of exchanged information
 - Coarse-grained
 - Fine-grained

As **granularity** **increases**, **flexibility** **decreases**

**Granularity is a design decision
with design tradeoffs**



Business Architecture



SOA

- Platform Independent
- Focused developer roles
- Loosely coupled
- Reusability
- Cost reduction
- Scalability
- Availability

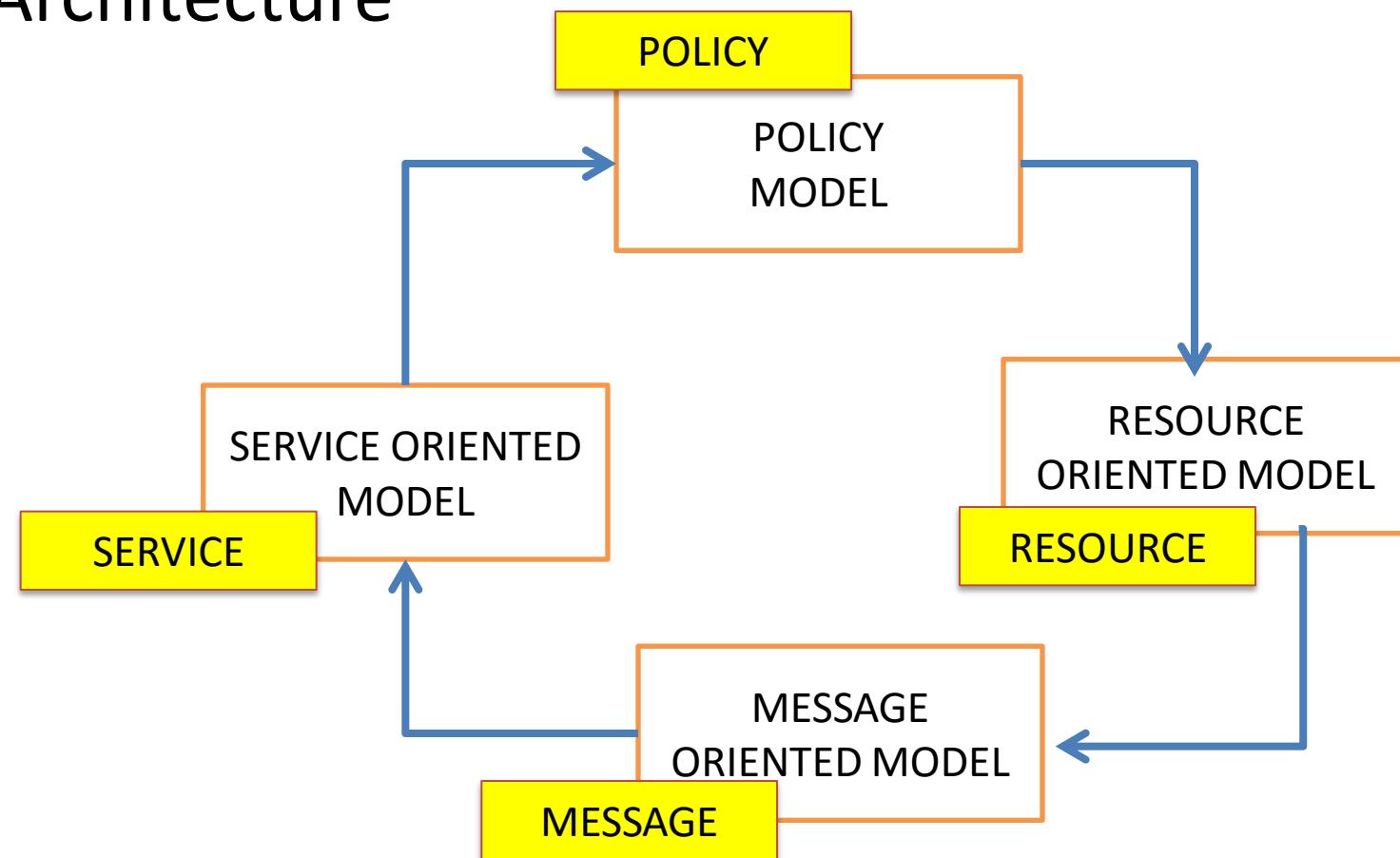
SERVICE IDENTIFICATION

- REFINEMENT
 - SERVICES COULD BE MERGED
 - NEW CAPABILITIES COULD BE ADDED
 - CAPABILITIES COULD BE MERGED
- IMPLEMENTATION AGNOSTIC
 - SOAP VS. WEB APIs
 - SOAP: Services and Operations
 - WEB APIs: Resource and Actions.

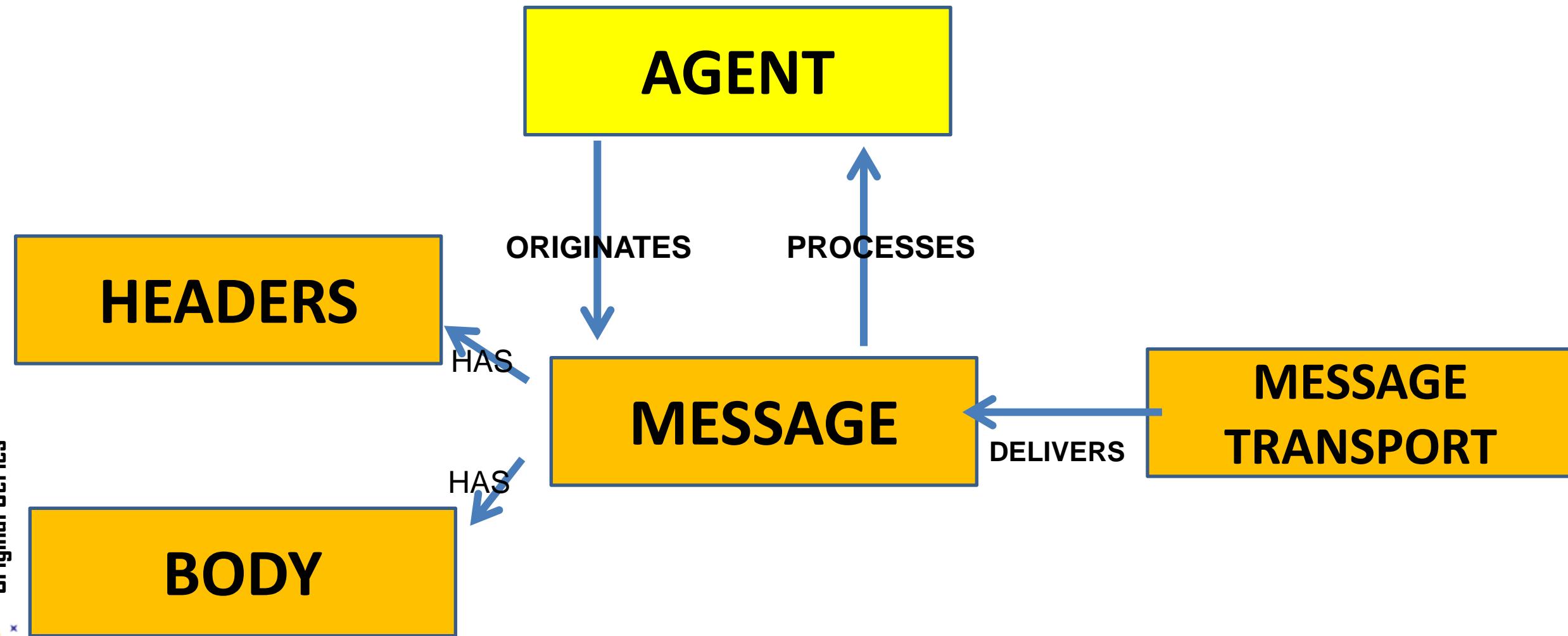


Foundations of Web Services

- 4 views of Web Services Architecture
 - Message Oriented Model
 - Service Oriented Model
 - Resource Oriented Model
 - Policy Model



MESSAGE ORIENTED MODEL



MESSAGE ORIENTED MODEL

https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#message_oriented_model

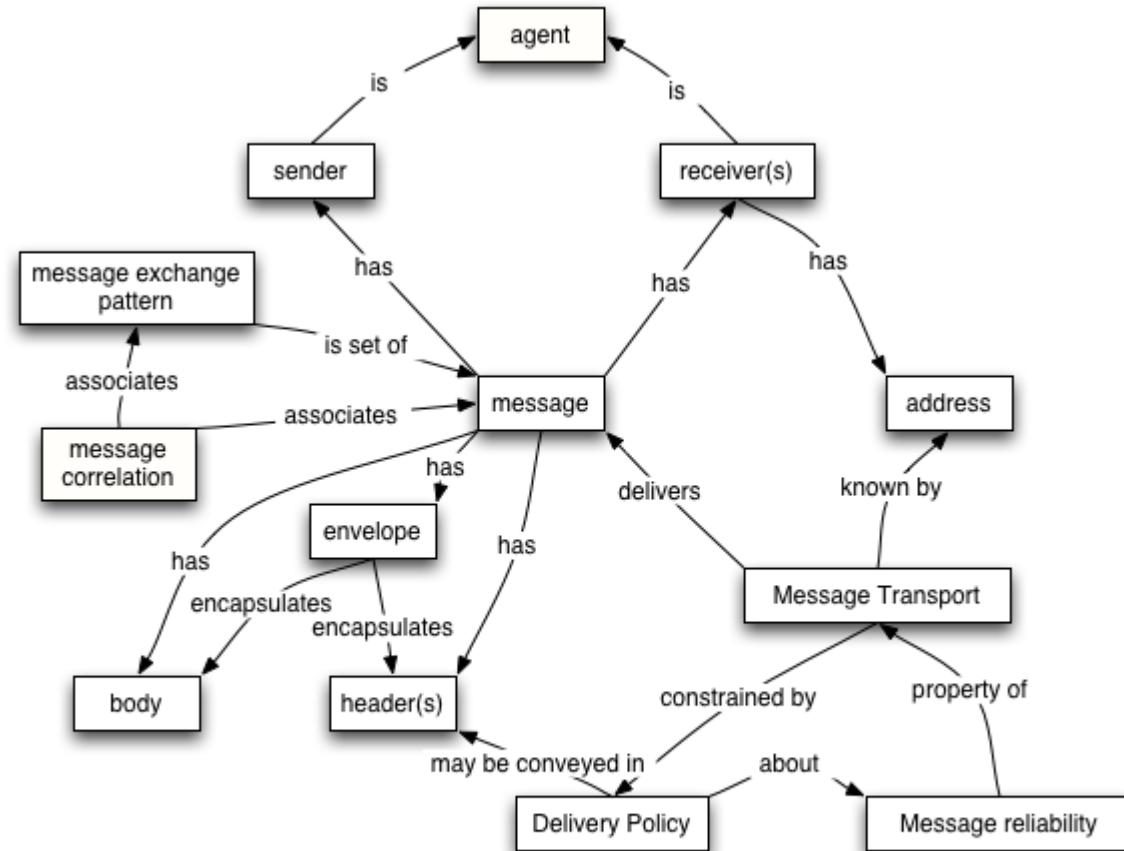


Figure 2-7. Message Oriented Model



SERVICE ORIENTED MODEL

https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#service_oriented_model

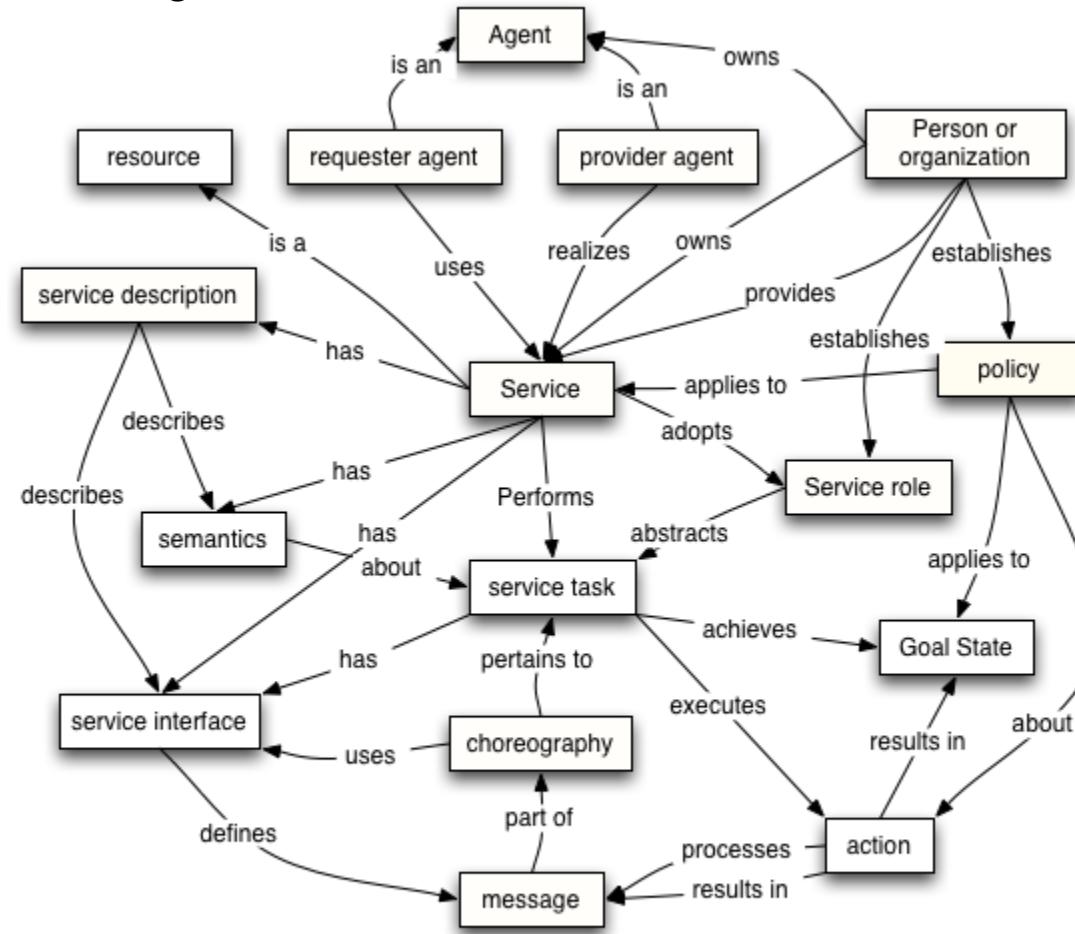
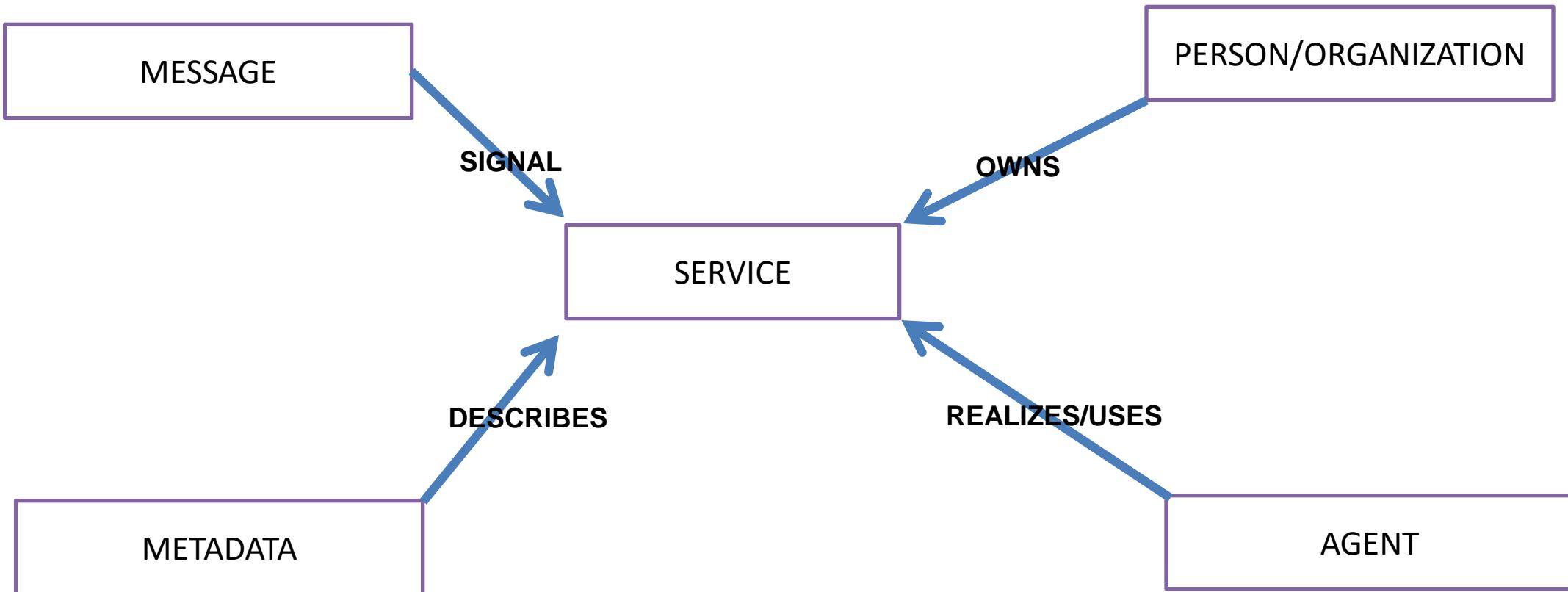


Figure 2-8. Service Oriented Model



SERVICE ORIENTED MODEL



RESOURCE ORIENTED MODEL

https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#resource_oriented_model

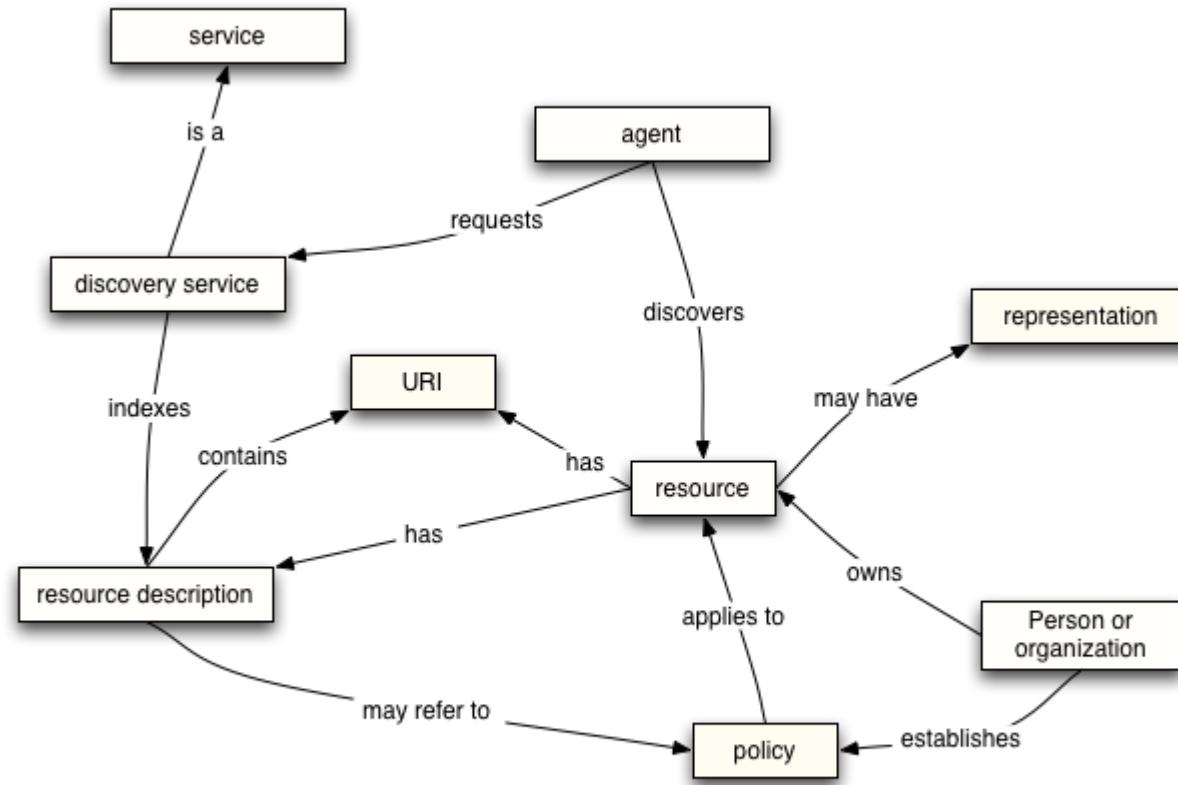
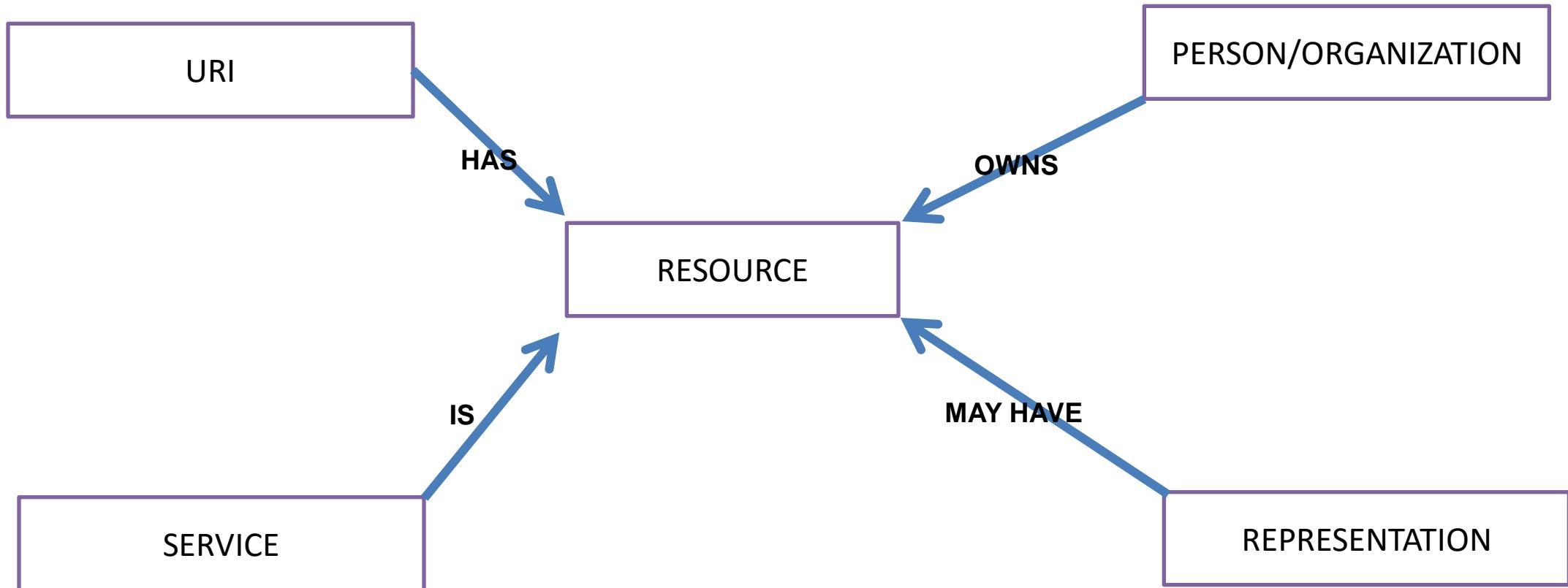


Figure 2-9. Resource Oriented Model

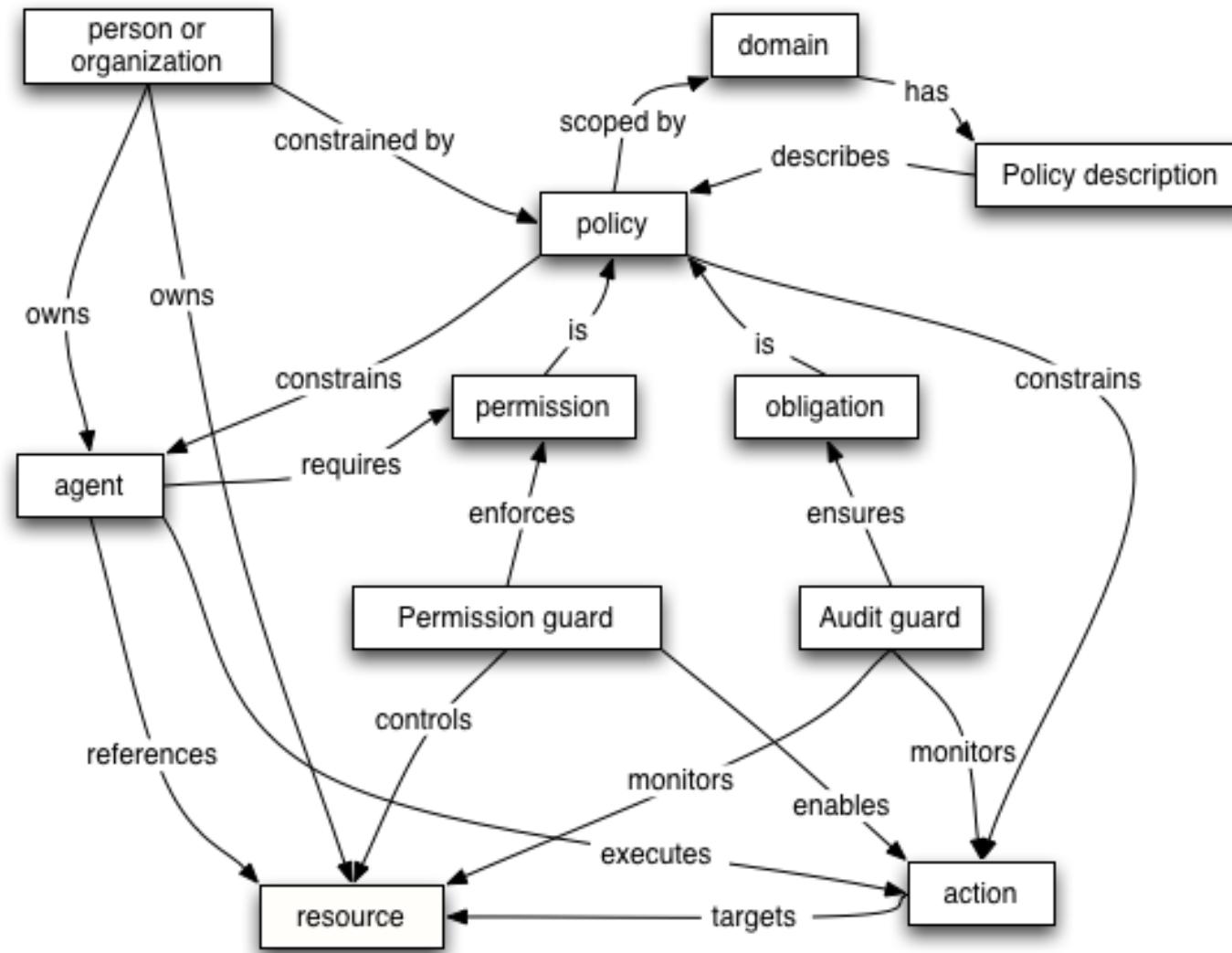


RESOURCE ORIENTED MODEL

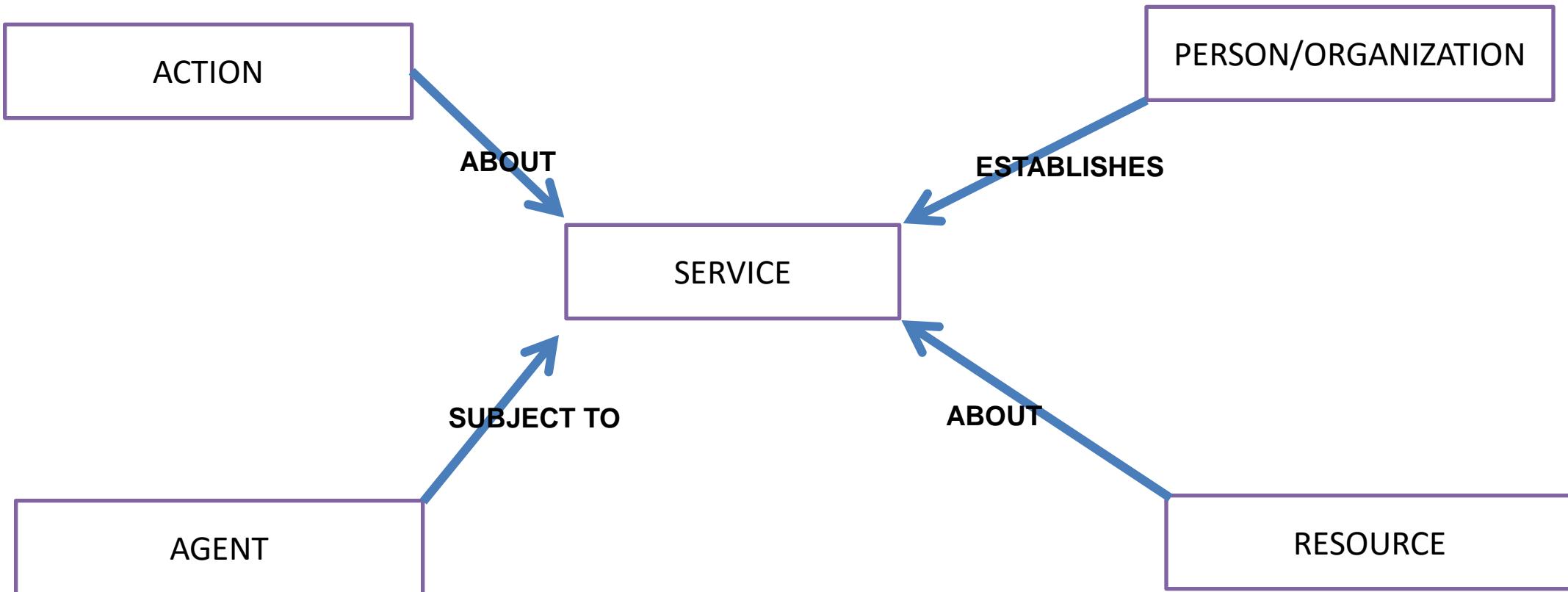


https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#policy_model

POLICY ORIENTED MODEL



POLICY ORIENTED MODEL



W3C-STYLE WEB SERVICES

MANAGEMENT

SECURITY

PROCESSES

BPEL

WS-CDL

CONTRACT

WSDL

WS-POLICY

MESSAGES

SOAP MESSAGES

SOAP

COMMUNICATIONS (HTTP,SMTP,)

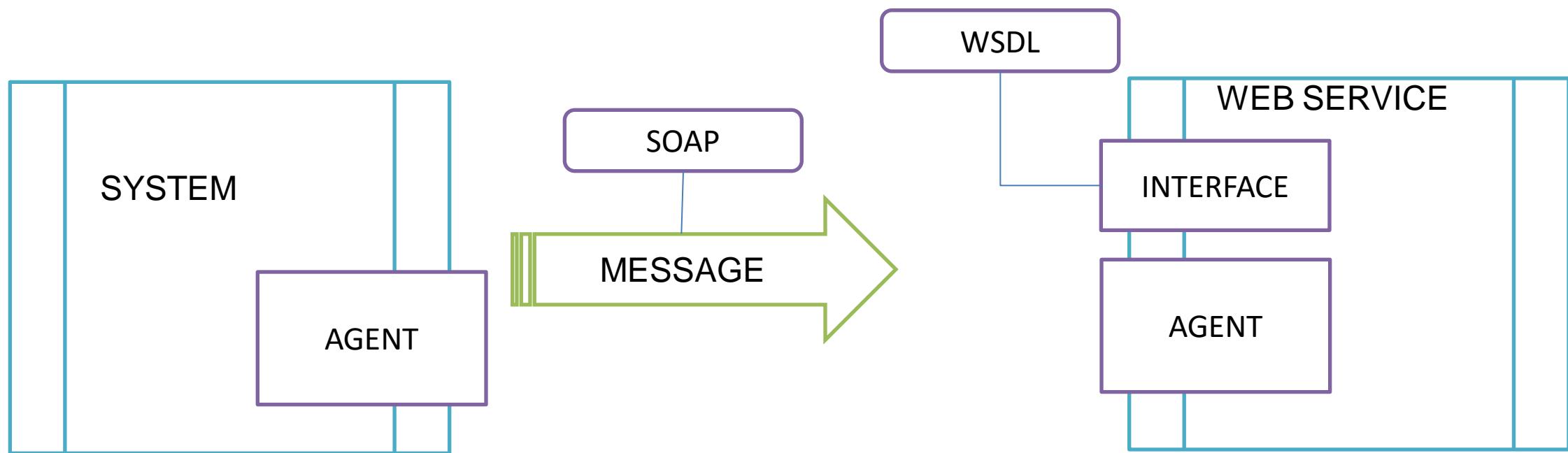
XSLT

XSD

XML



W3C-STYLE WEB SERVICES



SYED AWASE

IV: SOAP: SERVICES AND OPERATIONS

Version 1.2

SOAP

SOAP 1.1

- Released in May 2000
- Based on XML1.0 which is specific type of serialization
- Only supports HTTP
- Not compatible with SOAP 1.2
- Java implementations of SOAP are called SOAP with Attachment API for Java or SAAJ.

SOAP 1.2

- Released in June 2003
- Based on XML Infoset which provides a way to describe an XML document but does not necessarily serialize the document with XML 1.0. As a result, SOAP 1.2 can use other serialization formats such as a binary protocol format, which can be more compact than a SOAP 1.1 message
- Supports any protocol that conforms to its framework (HTTP,TCP/IP and MQ)
- It has slightly more specific definition of the SOAP processing model which reduces the chance for compatibility problems between different vendors that use different SOAP 1.2 implementations
- Java implementations of SOAP are called SOAP with Attachment API for Java or SAAJ.
- Support for International Standards



Inter-Application Communication

RPC

- Remote Procedure Calls
- Designed for computers within a small network
- Blocked by some firewalls
- Language-independent

SOAP

- Simple Object Access Protocol
- Designed for remote communication
- Universally recognized
- Language-independent



Inter-Application Communication

CORBA/DCOM/JavaRMI

- ORPC protocols attempted to marry object orientation and network protocols.
 - Requires both ends of communication link to be implemented under the same distributed object model (Java RMI or CORBA/IOP)
 - Difficulty of getting these protocols to work over firewalls or proxy servers

SOAP

- Simple Object Access Protocol is an XML based communication protocol for exchanging messages between computers regardless of their operating systems, programming environment or object model framework.



WHEN TO USE SOAP?

- Formal contract is required (WSDL)
- Non-functional requirements
 - Security
 - Transaction management
- Reliable asynchronous processing



Simple Object Access Protocol

- Protocol for inter-application communication
 - Applications communicate in a decentralized and distributed environment.
 - SPIRIT Project (www.geo-spirit.org) used SOAP for inter-application communication between distributed applications hosted at different geographic research centers.
Alternatively REST would have been better choice.
- SOAP message is an XML document with envelopes which exchange data
- Messages are transferred via HTTP/FTP/SMTP/TCP protocol
- A de facto standard protocol for communication with web services.
- Easily extensible
- Overcomes differences among proprietary heterogeneous peers
- **Stateless, one-way message exchange paradigm**
- More complex communication patterns can be created
 - Request/response
 - Publish/subscribe



SOAP?

- A message format for one-way communication describing how a message can be packaged into an XML document.
- A description of how a SOAP message should be transported using HTTP(for web-based interaction) or SMTP(for e-mail-based interaction)
- A set of rules that must be followed when processing a SOAP message and simple classification of the entities involved in processing a SOAP message
- A set of conventions on how to turn an RPC call into a SOAP message and back.



SOAP ELEMENTS

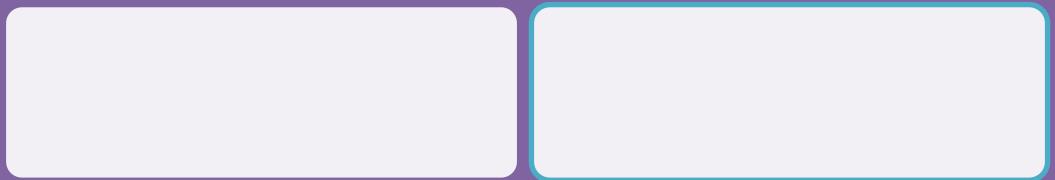
SOAP ENVELOPE

HEADER (OPTIONAL)

STATUS
CODE

ERROR-
CHECKING

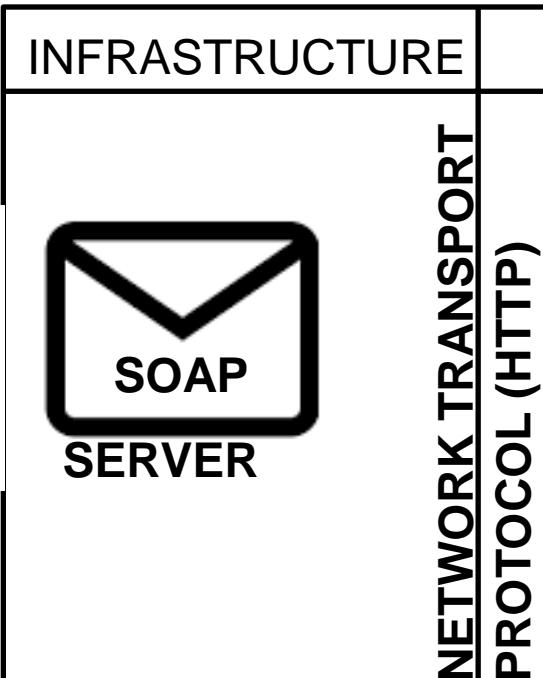
BODY



SOAP Messaging

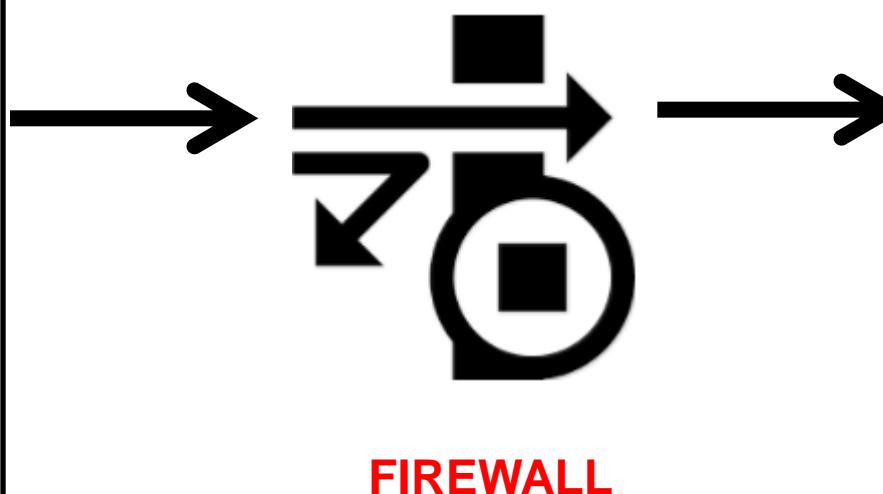
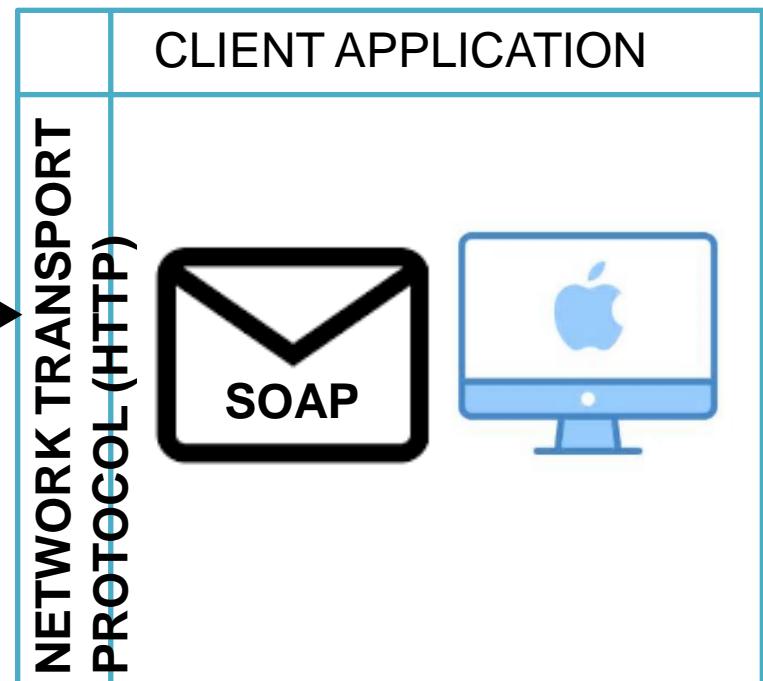
WEB SERVICE PROVIDER

SERVER

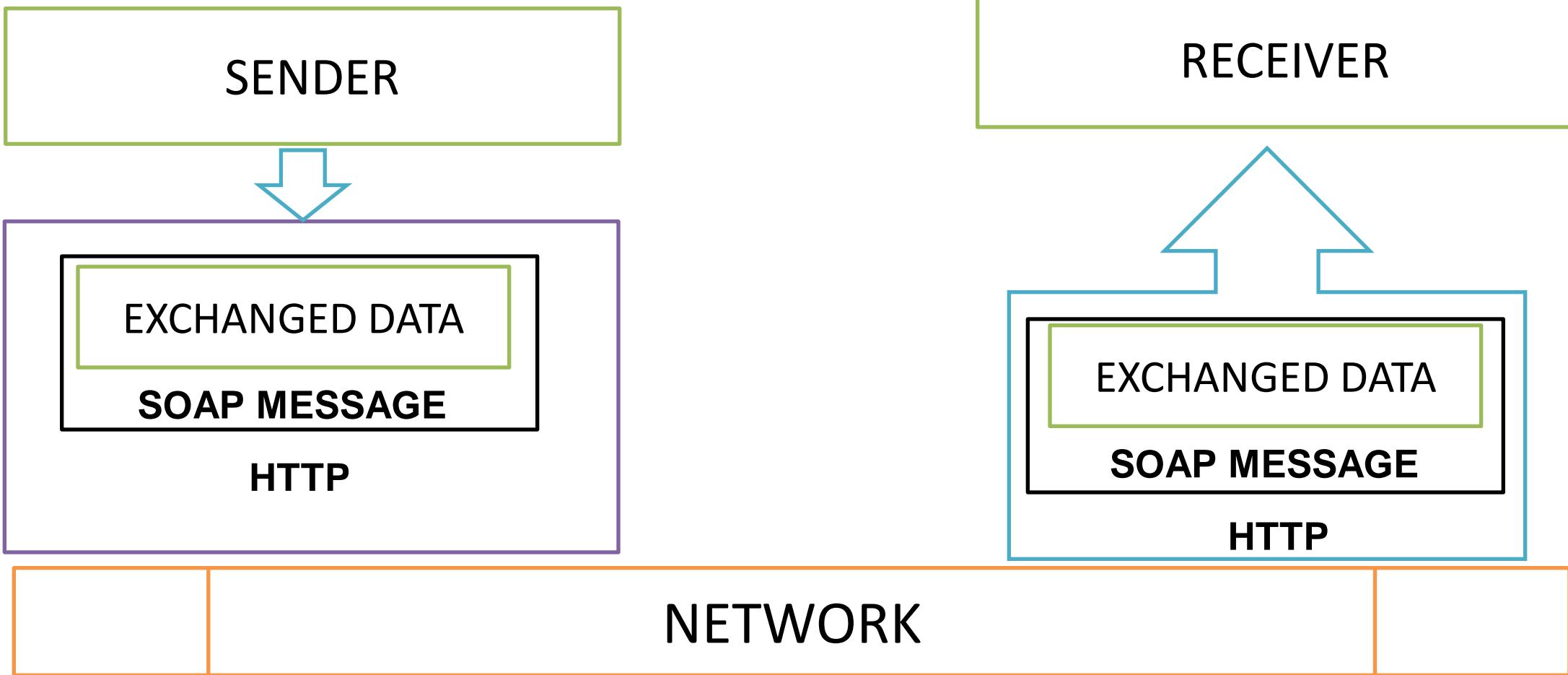


WEB SERVICE REQUESTOR

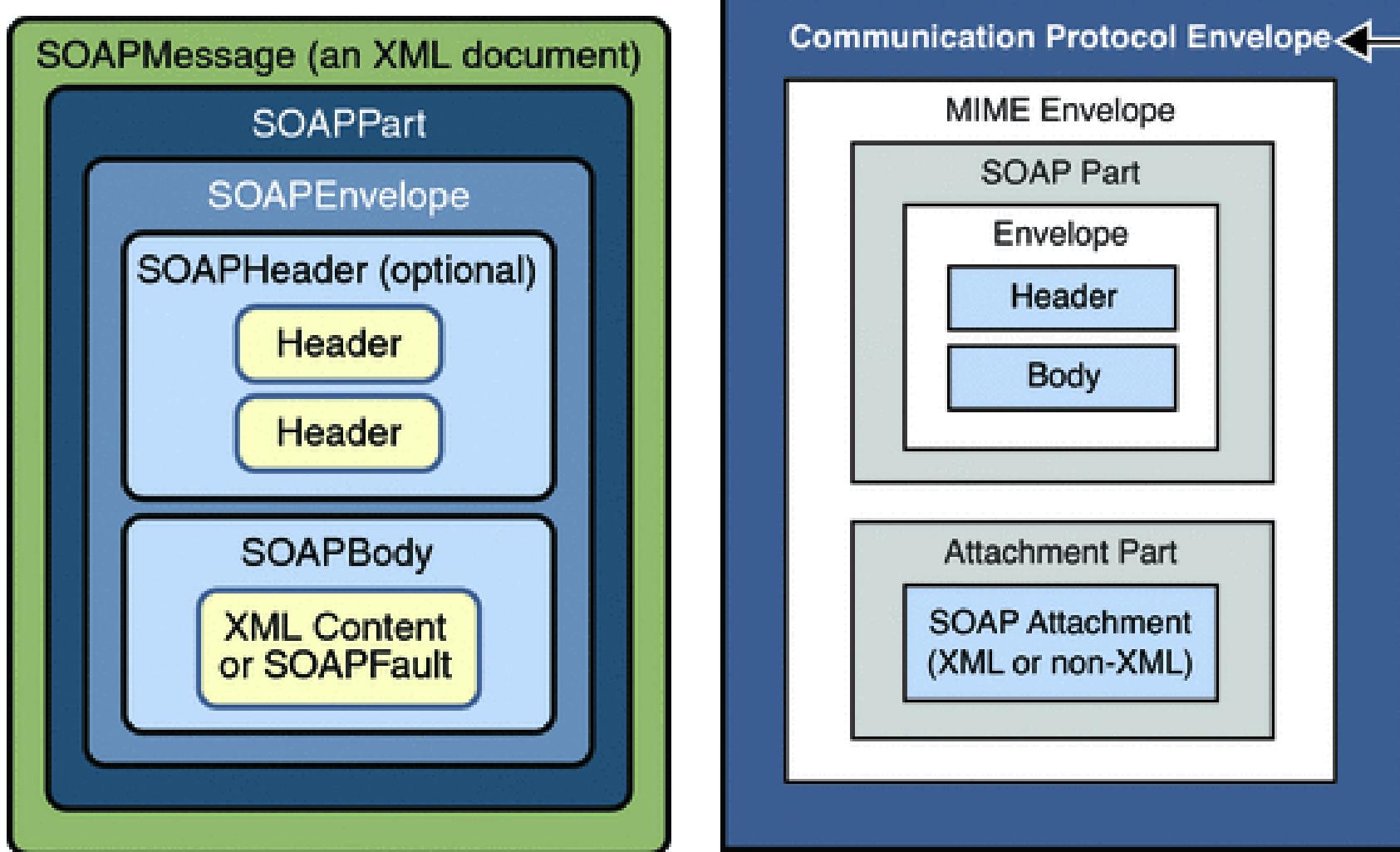
CLIENT APPLICATION



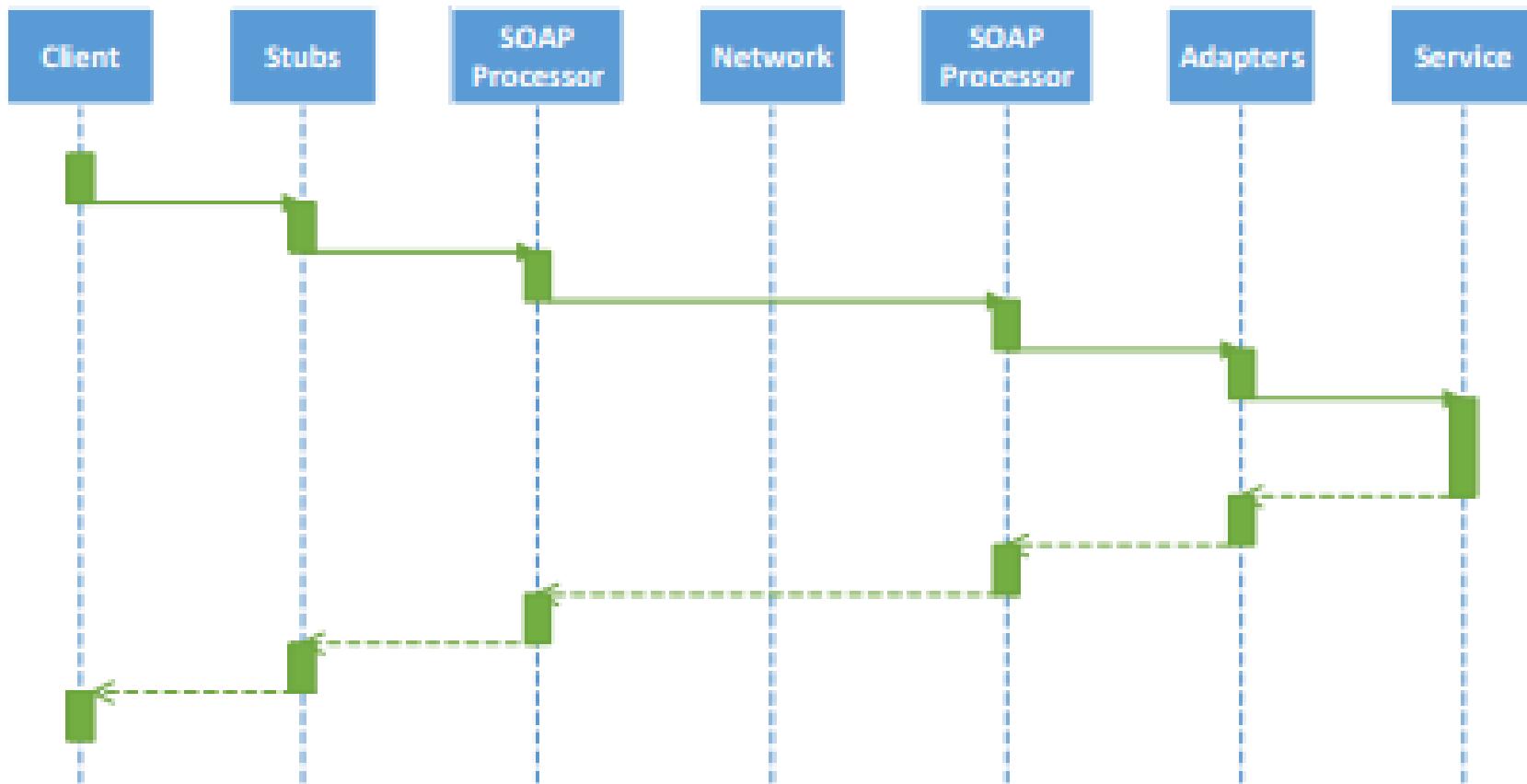
SOAP MESSAGE SYNTAX



SOAP



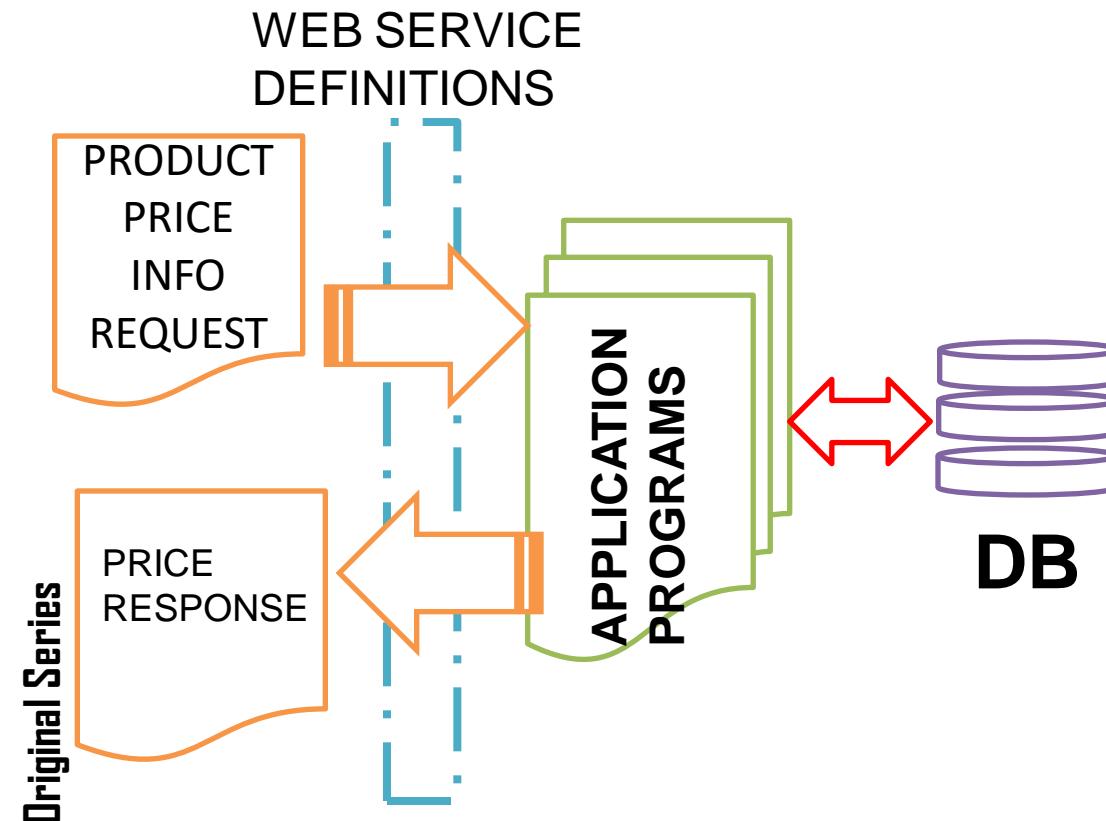
SOAP SEQUENCE DIAGRAM



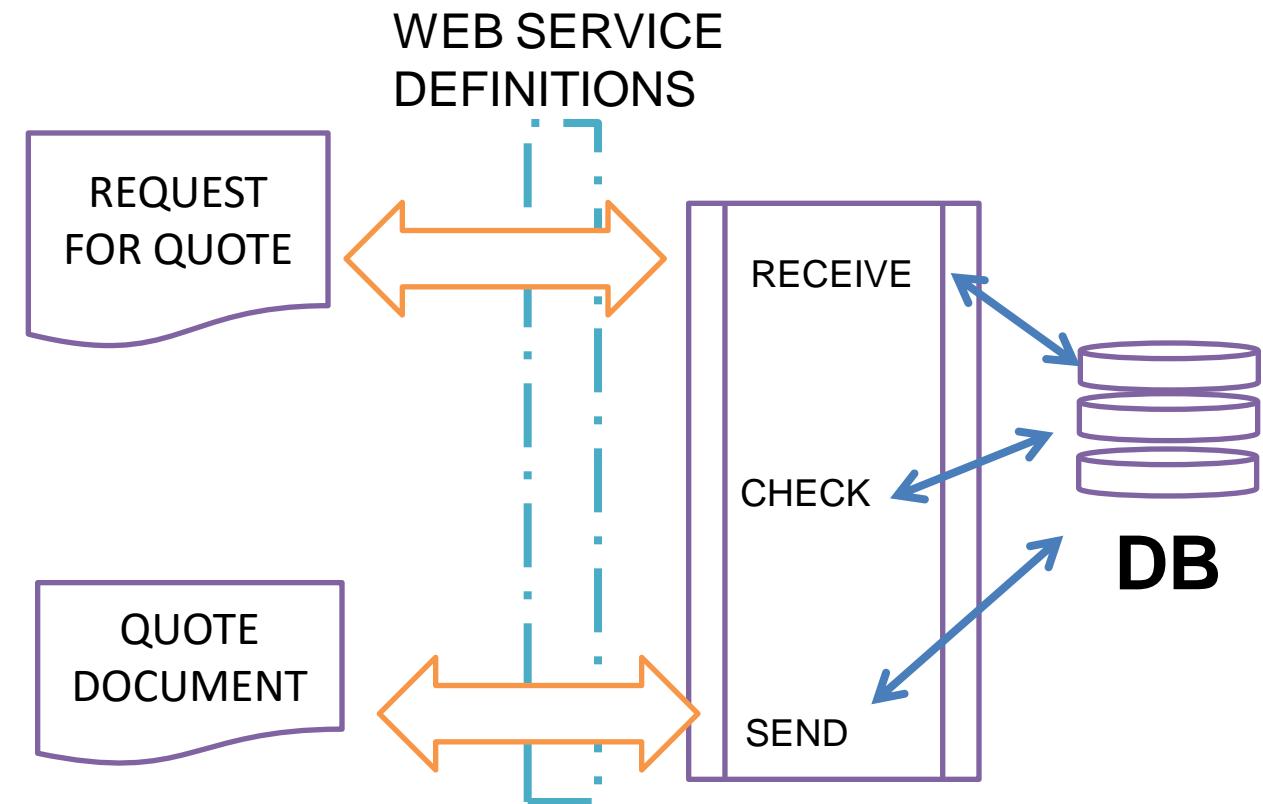
SOAP COMMUNICATION MODEL

SUPPORTS TWO POSSIBLE COMMUNICATION STYLES:

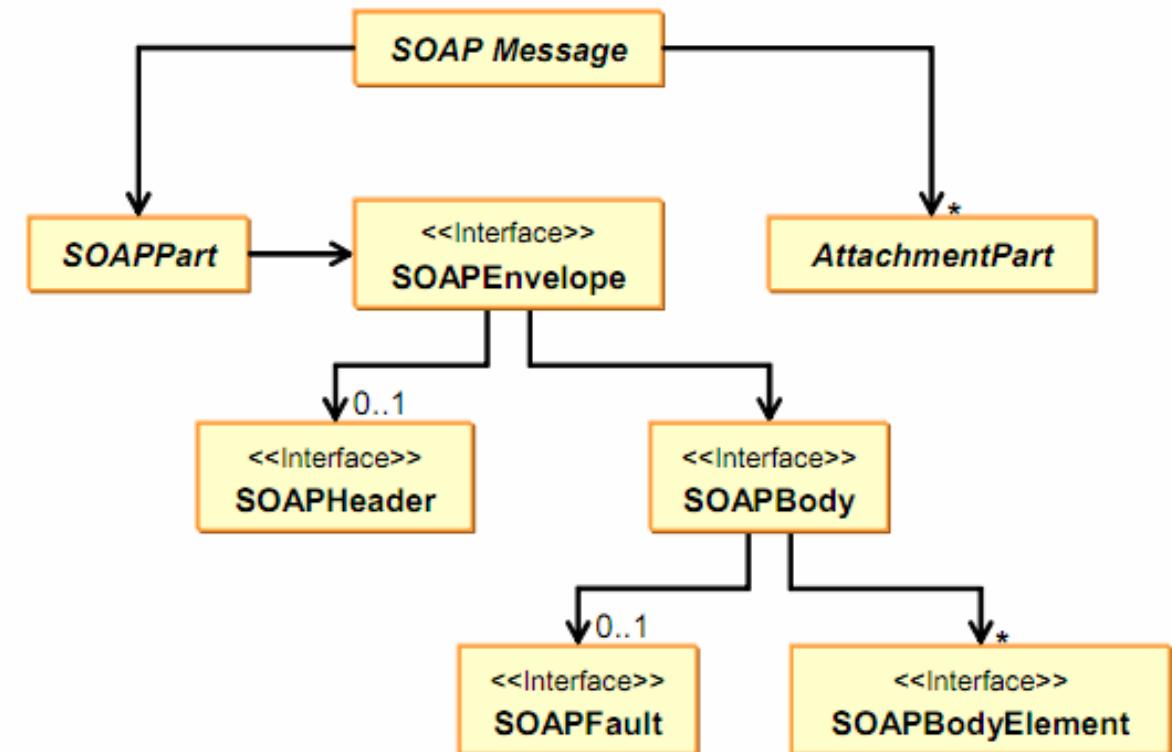
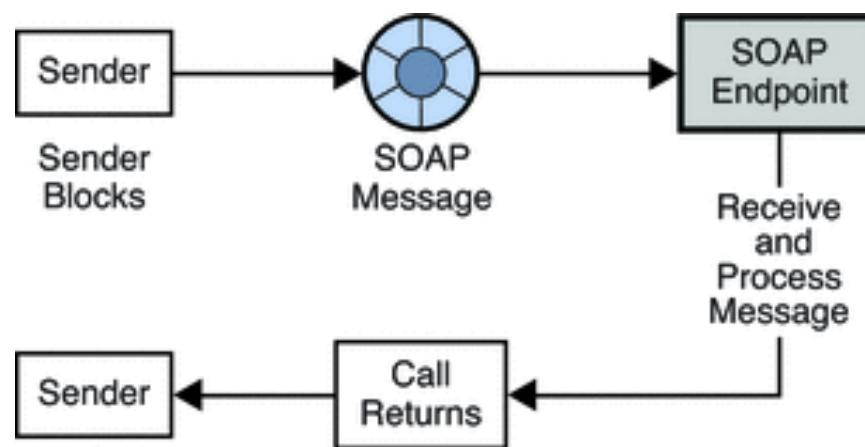
REMOTE PROCEDURE CALL (RPC)



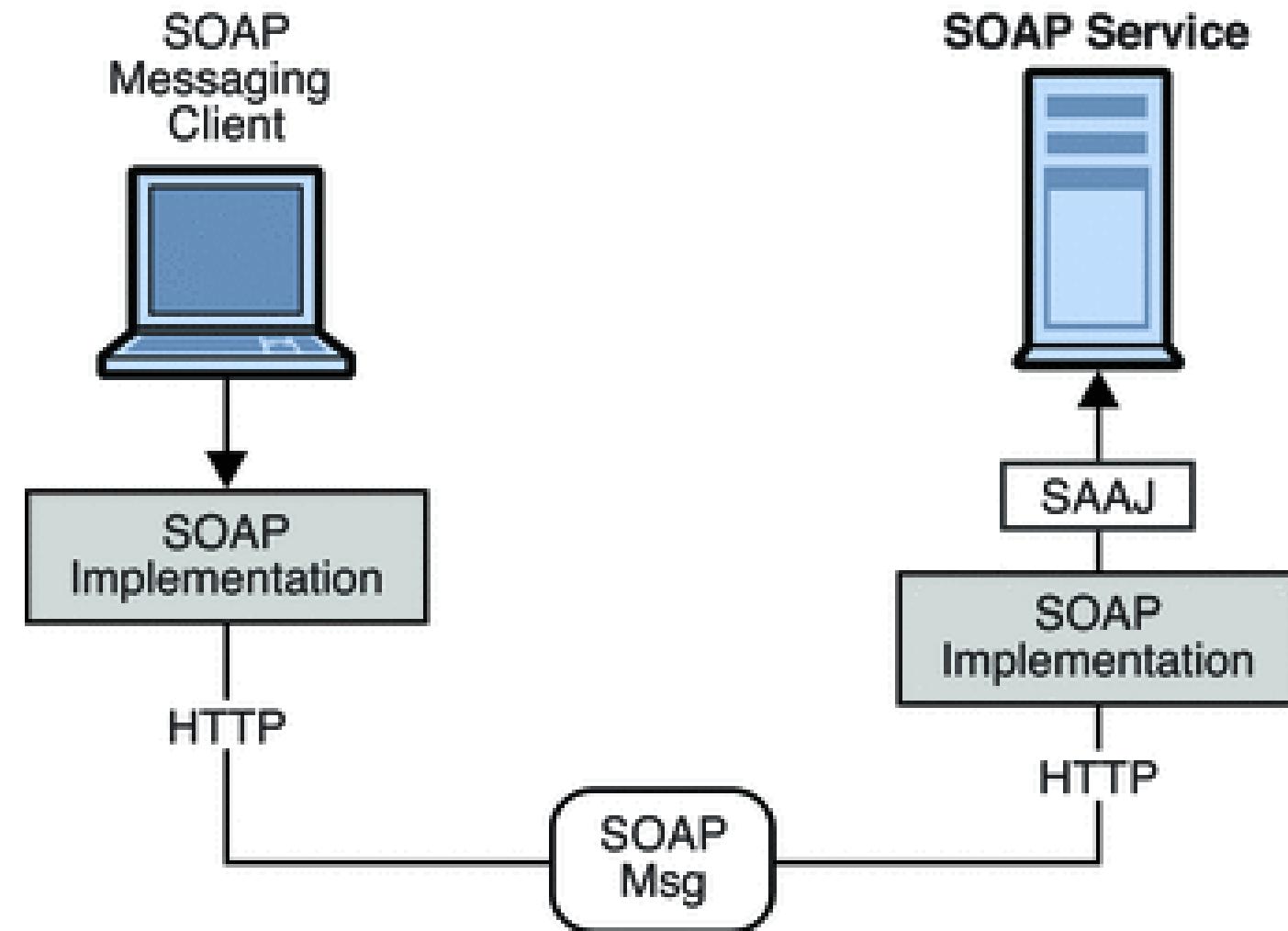
DOCUMENT(MESSAGE STYLE)



SOAP



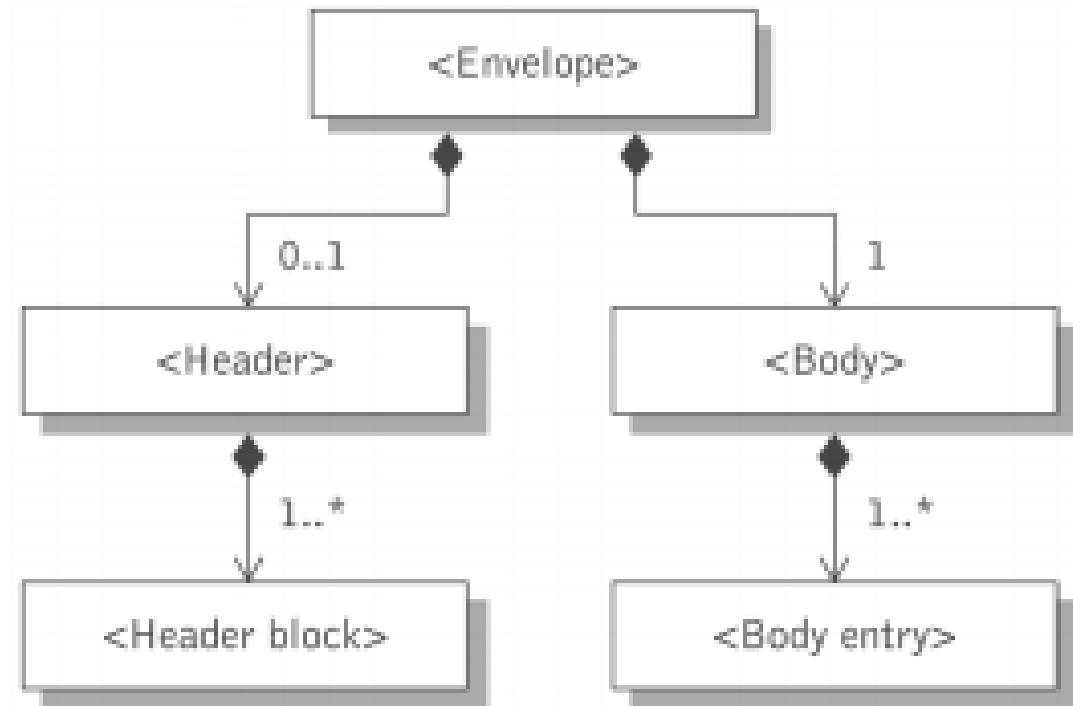
SOAP



SOAP MESSAGE SYNTAX

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
    SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
        <!--Header is optional-->
        <SOAP-ENV:Header>
            ...
            <!--one or more header blocks-->
        </SOAP-ENV:Header>

        <!--Body is mandatory-->
        <SOAP-ENV:Body>
            <!--Body is mandatory-->
            <SOAP-ENV:Fault>
                ...
                ...
            </SOAP-ENV:Fault>
            ...
        </SOAP-ENV:Body>
    </SOAP_ENV:Envelope>
```



SOAP:HEADER (OPTIONAL)

- Application specific XML elements
- Carries data that is not part of the application itself.
 - E.g: meta-data information (message addressing, security, transactions)
- SOAP extension mechanism
 - SOAP modules
 - SOAP faults
- SOAP extensions via header blocks:
 - Specific languages extending SOAP
 - WS-encryption, WS-Trans, WS-Addressing
 - W3C, OASIS
- Each header block should have its own namespace
 - Helps identify relevant header blocks



SOAP:BODY

- Application specific XML elements
- Carries application data
 - End-to-end information
- It may contain a number of child elements called body entries, but it may also be empty.
- A body element contains either the application-specific data, fault message
- A SOAP message may carry either application-specific data or a fault, **but not both.**
- **Application-specific data is the information that is exchanged with a web service. It is where the response to a method call is placed and where error information can be stored.**
- **A fault message is used only when an error occurs.**



SOAP:FAULT

- It provides a model for handling faults arise.
- It distinguishes between the conditions that result in a fault and the ability to signal that fault to the originator of the faulty message or another node.

Error Checking	
SOAP 1.1	SOAP 1.2
<faultcode> : to uniquely identify the fault	<code>: fault code in SOAP 1.1 simplified to <code>
<faultstring>: error message seen by a human	<Reason>: error message seen by a human, additional multiple sub-codes, multiple text strings
<faultactor>: an identifier for the node that created the fault	<Node>: an identifier for the node that created the fault.
<detail>: application-specific error information related to the body element.	<detail>: remains same as in SOAP 1.1
	<role>: indicates the role that node was playing at the point the fault occurred



SOAP:Faults

- All SOAP-specific and application-specific faults are reported using single element **FAULT** in **BODY**
 - network transfer protocol faults are reported using other protocol-specific mechanisms(e.g.: HTTP)
 - Separate SOAP message
 - Mandatory **CODE** and **REASON**
 - **OPTIONAL: DETAIL, NODE, ROLE**
- **SOAP FAULT: CODE**
 - Reports specific kind of fault
 - Particular kind of fault may require additional header blocks to be generated
 - Mandatory **value**
 - **Contains code value**
 - Optional **Subcode**
 - **Contains mandatory value with application-specific sub-code value**



SOAP:FAULTS

CODE	SEMANTICS
VERSIONMISMATCH	Faulting node does not support the given version of SOAP. The node SHOULD specify how the messages should be upgraded with Upgrade header
MUSTUNDERSTAND	Faulting node does not understand a header. The node should specify which header was not understood with NotUnderstood header
DATAENCODINGUNKNOWN	Faulting node does not understand the message encoding
SENDER	The message was incorrectly formed or did not contain the appropriate information in order to succeed. E.g.: improper authentication details, registration information etc.
RECEIVER	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. Something went wrong.



SOAP and HTTP

- A binding of SOAP to a transport protocol is a description of how a SOAP message is to be sent using that transport protocol
- SOAP can use GET or POST. But **with GET the request is not a SOAP message but the response is a SOAP message.**
- **With POST method, both request and response are SOAP messages (in version 1.2 , version 1.1 mainly considers the use of POST)**
- **SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly interpreted by a SOAP module.**



SOAP

Advantages

- Simple to **some extent**
- Portability
- Firewall friendliness
- Use of open standards
- Interoperability
- Universal acceptance

Disadvantages

- Too much reliance on HTTP, a stateless protocol
- Statelessness
- Serialization by value and not by reference.



Why SOAP?

- Reliable and Realistic Messaging:
SOAP consists of an inbuilt messaging system that helps in creating an end-to-end messaging system for clients.
 - REST does not hold any standard messaging system that offers no automation system to request or receive information.
- Security:SOAP is compatible to the SSL and supports WS-Security that helps in adding more enterprise security feature.
- AUTO-TRANSACTION: completing ACID transactions on any service using SOAP



SOAP

- SOAP depends primarily on XML to provide message services. SOAP uses different protocols for communication, such as HTTP, SMTP or FTP.
- It uses only XML for messages.
- SOAP clients are tightly coupled with the server and the integration would break if a change is made at either end.



SOAP Technologies/Implementations

SOAP Implementations for Java API: SAAJ 1.3 version



<https://axis.apache.org>

Apache CXF™

<https://cxf.apache.org>

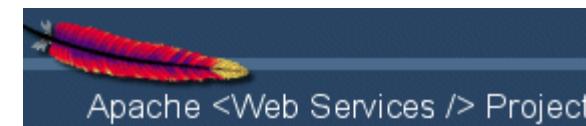


<http://jello-framework.com>



RESTful Web Services in Java.

<https://jersey.github.io>



<http://ws.apache.org/wsif/>

XINS

XML Interface for Network Services

<http://xins.sourceforge.net>

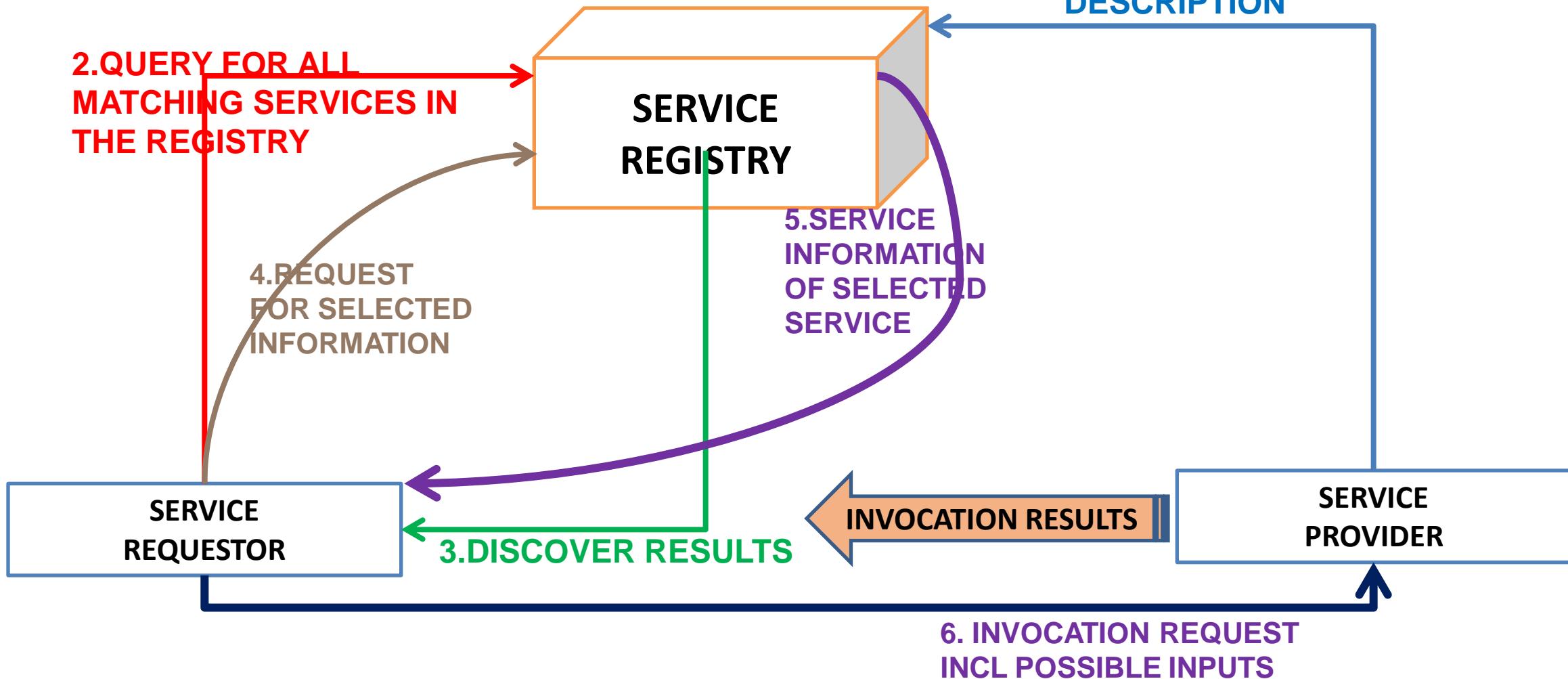




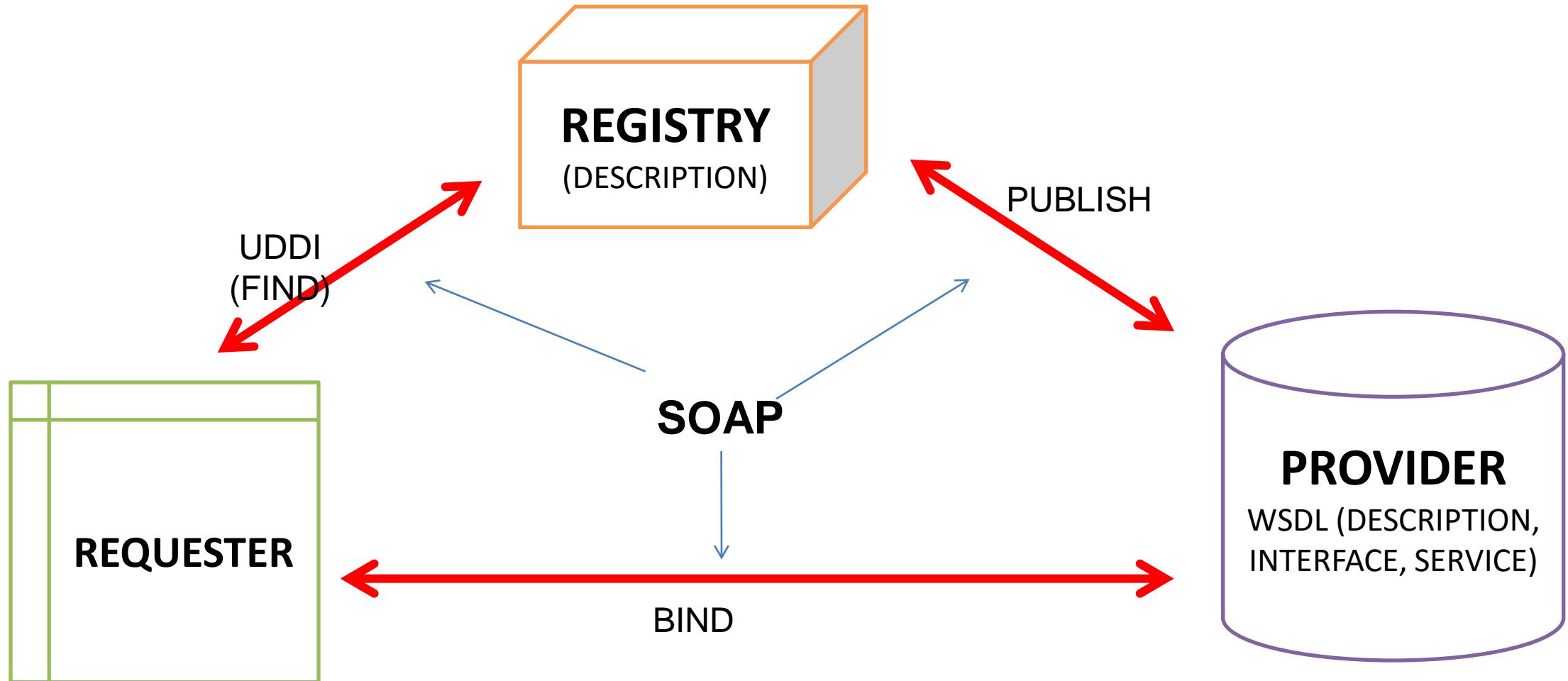
SYED AWASE

V: UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION

SOA Interactions



SOA INTERACTIONS



Service registries

- A web service registry provides description and registration of the web service
- Publication of a service requires proper description of a web service in terms of business, service and technical information.
- It also provides persistent storing the web service descriptions in the web services registry.
- Two types of registries
 - Document-based registry: enables its clients to publish information by storing XML-based service documents such as business profiles or technical specifications (incl. WSDL description of service)
 - Metadata based service registry: captures the essence of the submitted document.



Service discovery

- It is the process of locating web service providers, and retrieving web service descriptions that have been published.
- Interrogating services involve querying the service registry for web services matching the needs of a service requestor.
- A query consists of following criterion
 - Type of the desired service
 - Preferred price
 - Max no of returned results
 - Is executed against service information published by service provider.



Types of Service Discovery

- Static
 - Service implementation details are bound at design time and a service retrieval is performed on a service registry
 - Results of the retrieval operation are examined by human designer and the service description returned by the retrieval operation is incorporated into the application logic.
- Dynamic
 - The service implements details are left unbound at design time, so that they can be determined at run-time
 - The web service requestor has to specify preference to enable the application to infer/reason which web services the requestor is most likely to invoke
 - Based on application logic, quality of service considerations such as best price, performance, security certificates are invoked.



UDDI

- A standard for web service description and discovery together with a registry facility that supports the WS Publishing and Discovery process
- UDDI business registration consists of 3 inter-related components
 - White pages (address, contact and other key points of contact)
 - Yellow pages(classification info based on standard industry taxonomies)
 - Green pages – technical capabilities and information about services
- It enables a business to
 - Describe its business and its services
 - Discover other businesses that offered desired services
 - Integrate(interoperate) with these other businesses





SYED AWASE

VI: WEB SERVICES DESCRIPTION LANGUAGE

WSDL

- WSDL assists service clients that need to know how to bind a service automatically.
- WSDL is based on the Interface Description Language(IDL), which describes the interface of a software component for other components to use.
- It is used to create a “stub” which encodes the method invocation as SOAP
- WSDL structure consists of two main parts. It is an XML document with a root element named **definitions**
 - Abstract interface /Abstract descriptions
 - types
 - messages
 - portType
 - Concrete implementation.
 - Binding
 - service



Sample WSDL File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:co="http://www_tpri_com_cfdeomtwo/schema/CustomerOrder"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www_tpri_com_cfdeomtwo/CustomerOrders/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="CustomerOrders" targetNamespace="http://www_tpri_com_cfdeomtwo/CustomerOrders/"/>
<wsdl:types>
    <xsd:schema targetNamespace="http://www_tpri_com_cfdeomtwo/CustomerOrders/"/>
        <xsd:import namespace="http://www_tpri_com_cfdeomtwo/schema/CustomerOrder"
            schemaLocation="../xsd/CustomerOrder.xsd"></xsd:import>
    </xsd:schema>
</wsdl:types>
<wsdl:message name="NewOperationRequest">
    <wsdl:part element="tns:NewOperation" name="parameters" />
</wsdl:message>
<wsdl:message name="NewOperationResponse">
    <wsdl:part element="tns:NewOperationResponse" name="parameters" />
</wsdl:message>
<wsdl:portType name="CustomerOrders">
    <wsdl:operation name="NewOperation">
        <wsdl:input message="tns:NewOperationRequest" />
        <wsdl:output message="tns:NewOperationResponse" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CustomerOrdersSOAP" type="tns:CustomerOrders">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="NewOperation">
        <soap:operation
            soapAction="http://www_tpri_com_cfdeomtwo/CustomerOrders/NewOperation" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CustomerOrders">
    <wsdl:port binding="tns:CustomerOrdersSOAP" name="CustomerOrdersSOAP">
        <soap:address location="http://www.example.org/" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

} XSD SCHEMA

} Request and Response Message mapping

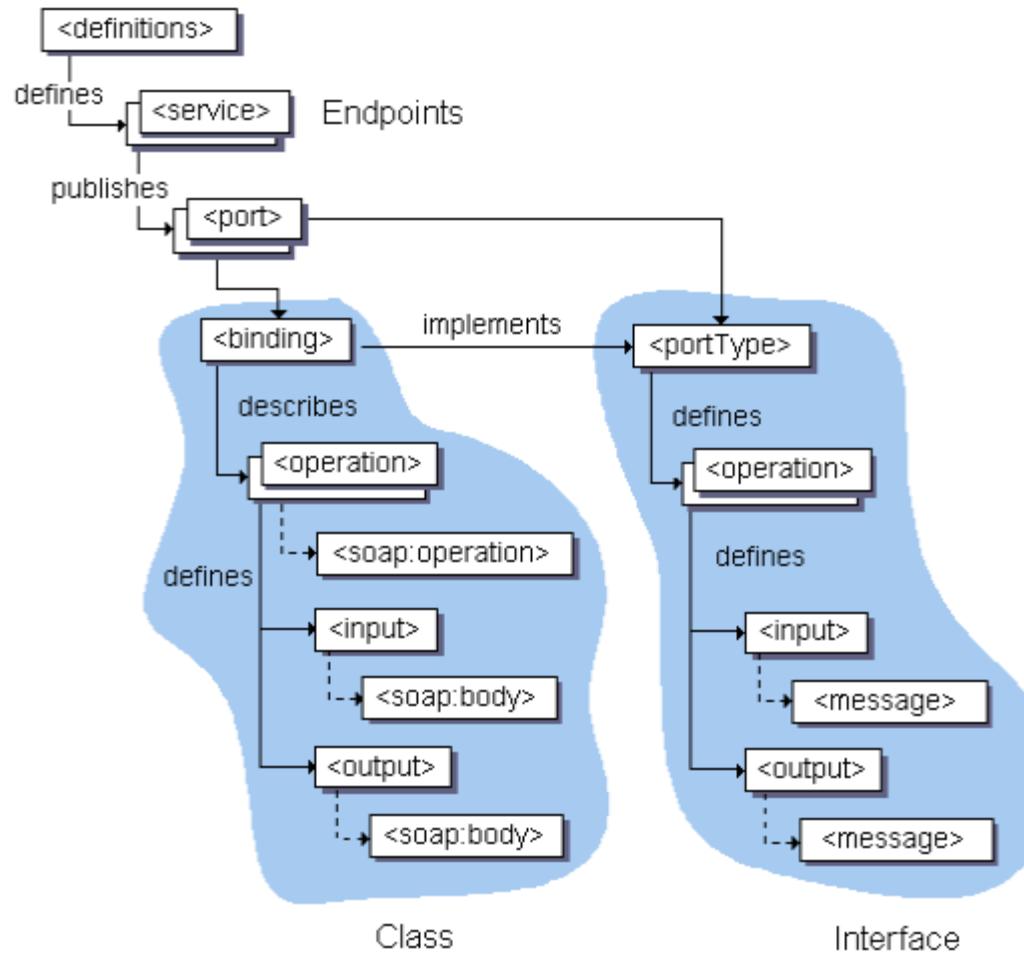
} Port

} Bindings

} Service Endpoint Interfaces



Structure of a WSDL Document



- WSDL document has **definitions** element that contains the other 5 elements
 - **Types**
 - **Messages**
 - **portType**
 - **Binding**
 - **Service**

Source: http://download.oracle.com/otn_hosted_doc/jdeveloper/1012/web_services/ws_wsdlstructure.html



WSDL DEFINITIONS

- Contains the definition of one or more services.
 - Name is optional
 - targetNamespace : logical namespace for information about this service.
 - xmlns:default namespace of the wsdl
 - All the wsdl elements such as **definitions, types and message reside in this namespace.**
- XmlNs:xsd and xmlNs:soap are standard namespace definitions that are used for specifying SOAP specific information as well as data types.
- XmlNs:tns (target namespace) stands for this namespace.



WSDL Types

- Provides information about any complex data types used in the WSDL document.
- When simple types are used the document does not need to have a types section.
- A container for data type definitions using some type system such xsd



WSDL Message

- An abstract definition of the data being communicated.
- A message contains both
 - Request
 - Response



WSDL Operation

- An abstract description of the action supported by the service.



WSDL PORTTYPE

- An abstract set of operations supported by one or more endpoints.



WSDL BINDING

- It describes how the operation is invoked by specifying concrete protocol and data format specifications for the operations and messages.
- A concrete protocol and data format specification for a particular port type.



WSDL PORT

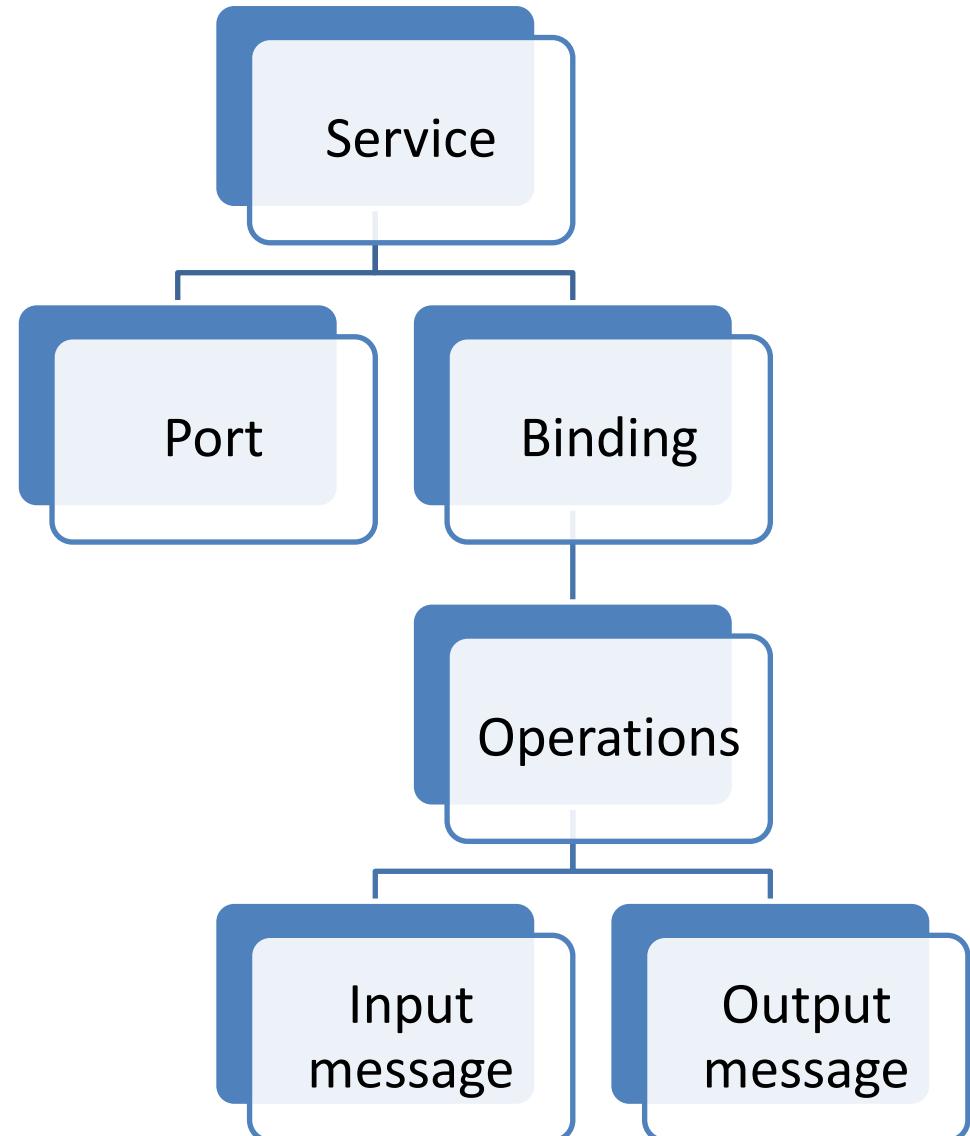
- It specifies a single end point as an address for the binding, thereby defining a single communication endpoint.



WSDL SERVICE

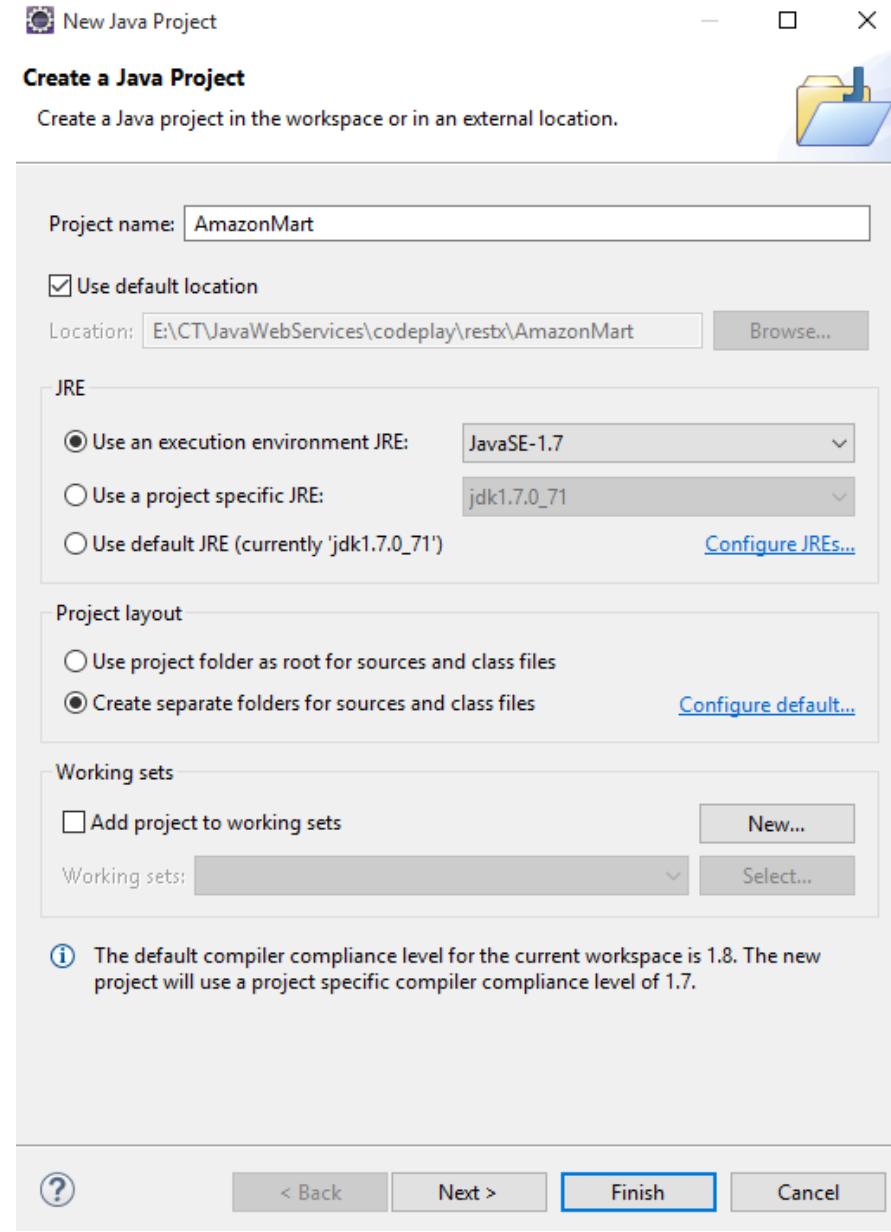
- It specifies the port address(es) of the binding. The service is a collection of network endpoints or ports.





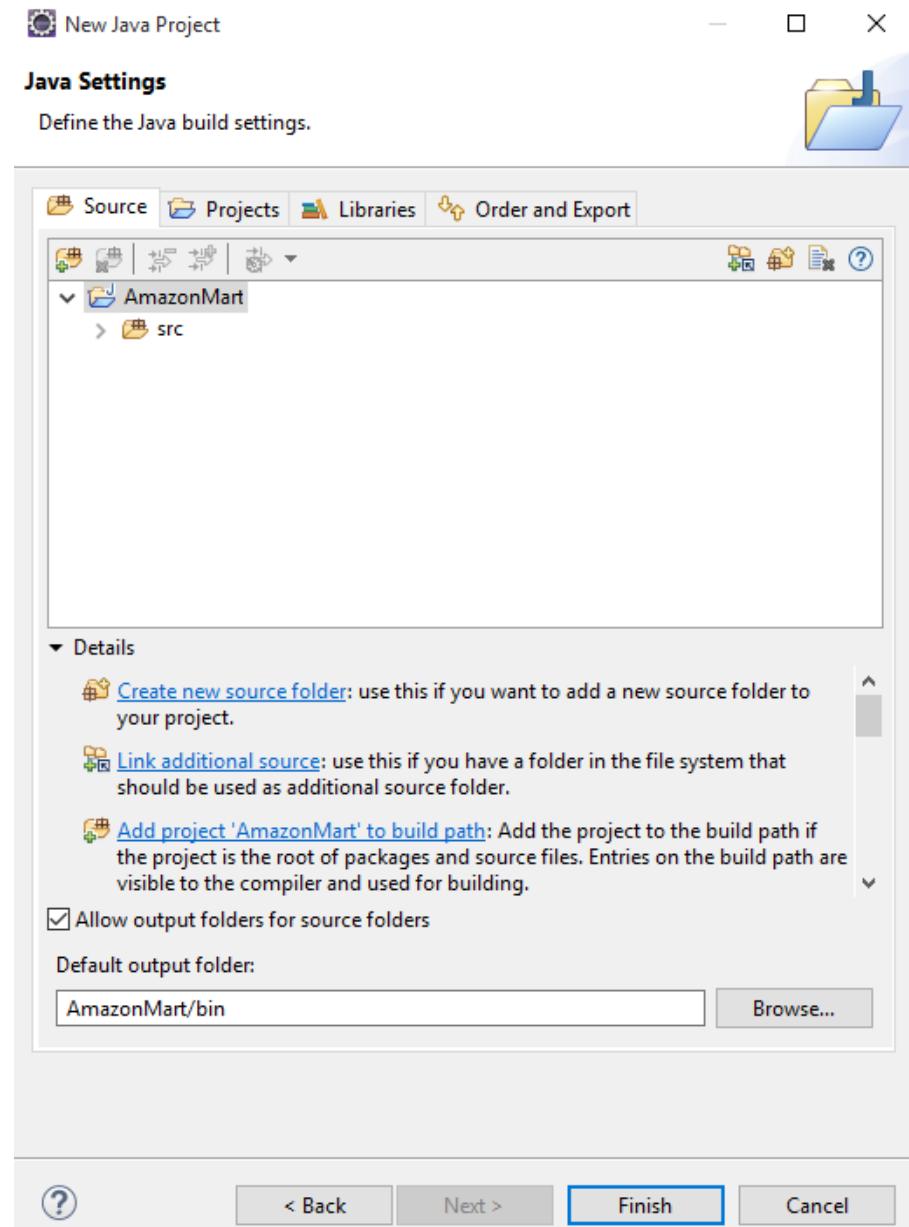
Step 1

- create a Java Project => AmazonMart



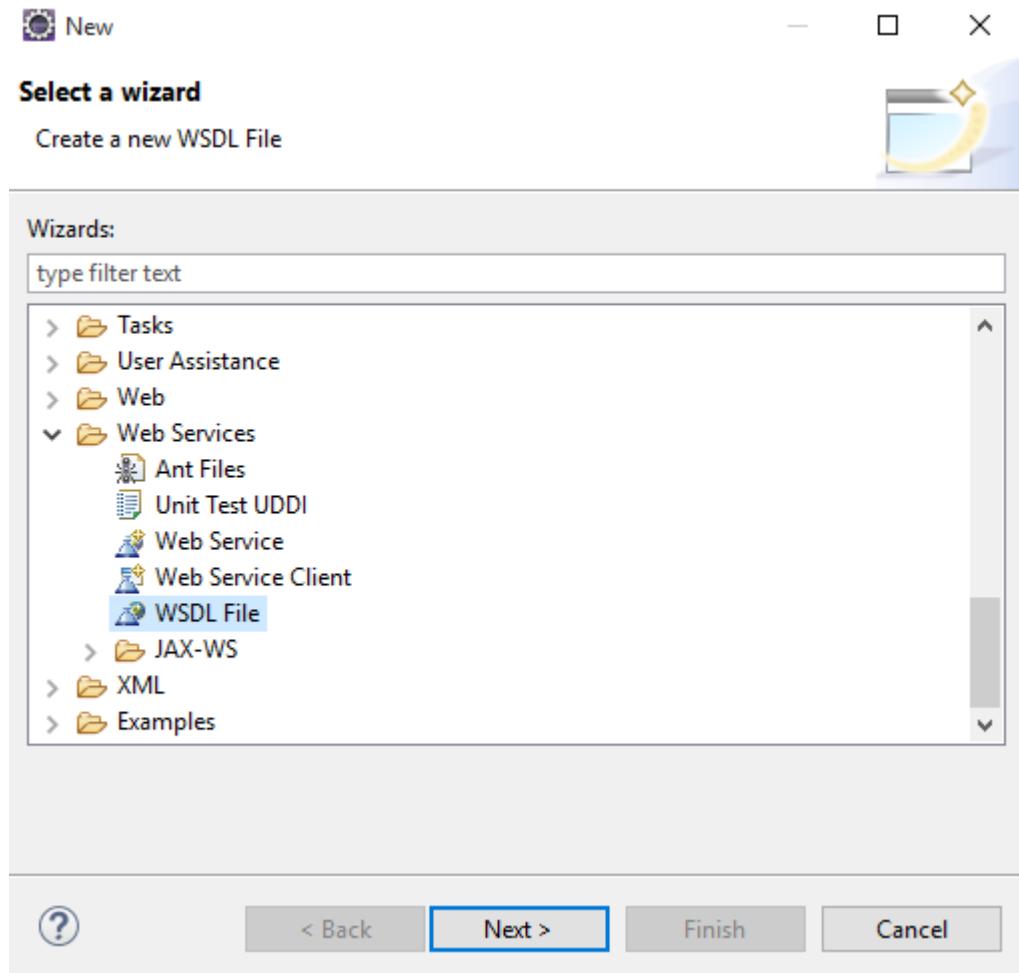
Step 2

-



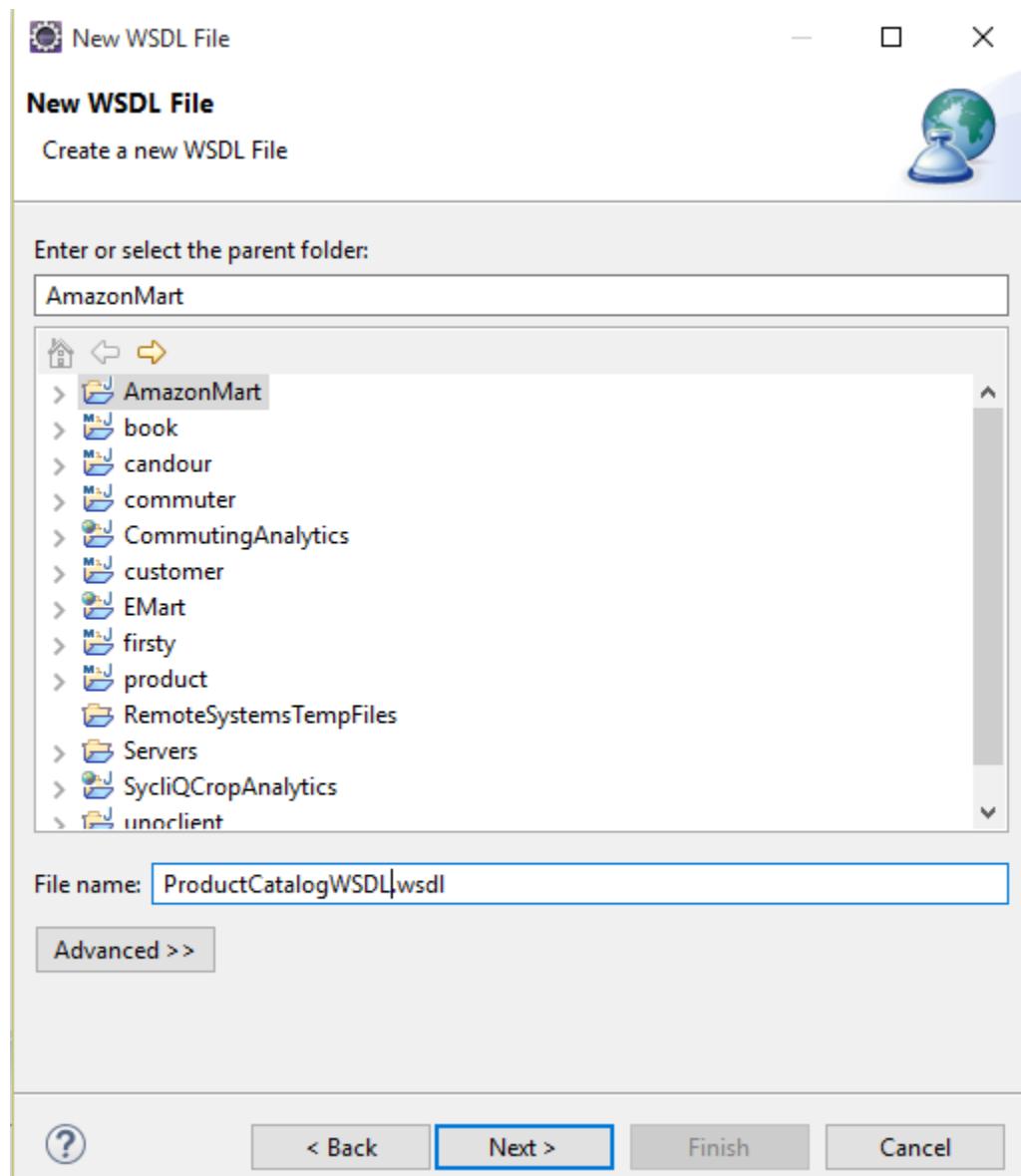
Step 3

- create a new WSDL file



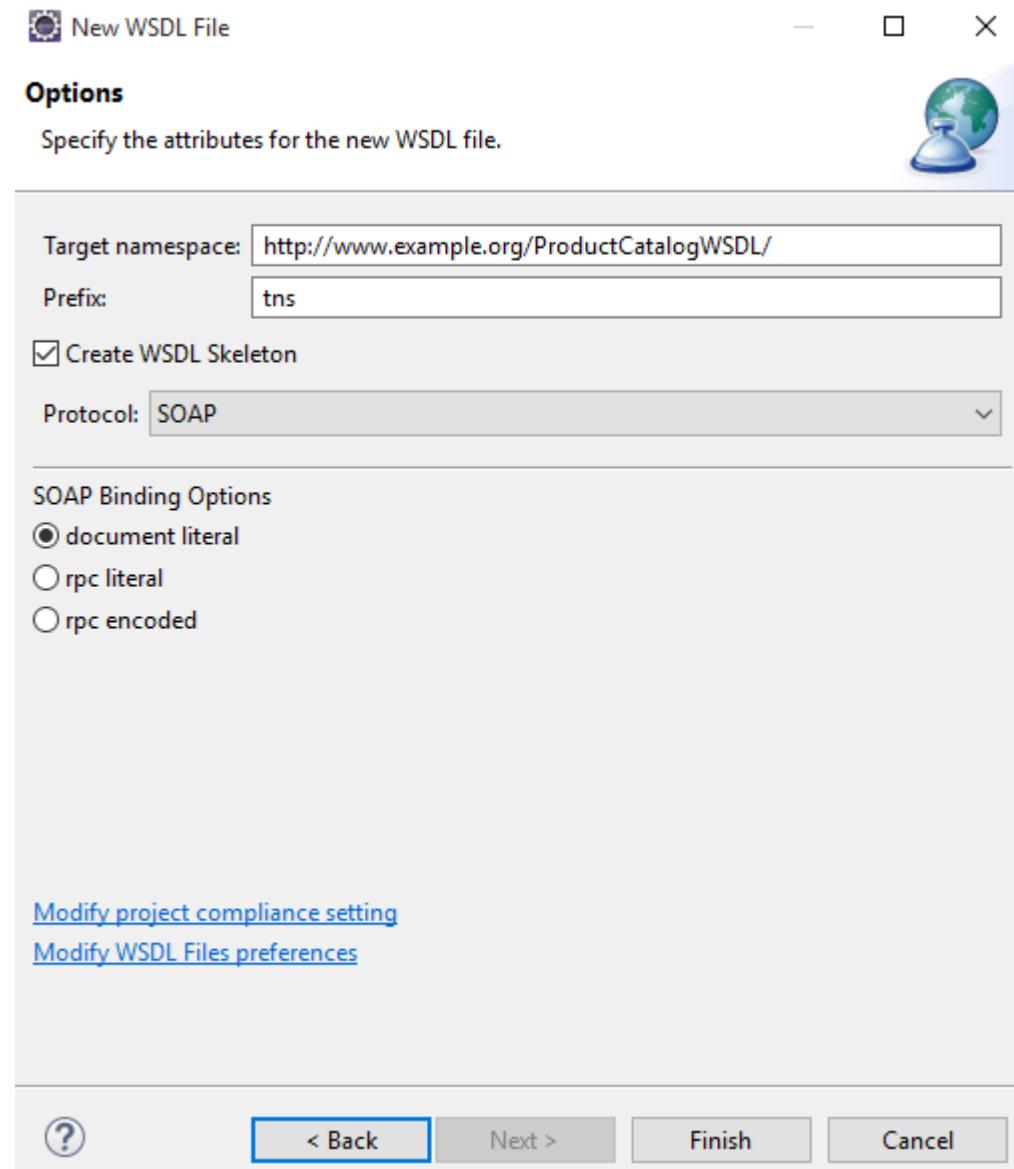
Step 4

- Create
ProductCatalogWSDL.wsdl
file



Step 5

- Set the target namespace
- select the WSDL skeleton with **SOAP PROTOCOL OPTION**
- select **SOAP Binding Options**
- **Document literal**



Step 6

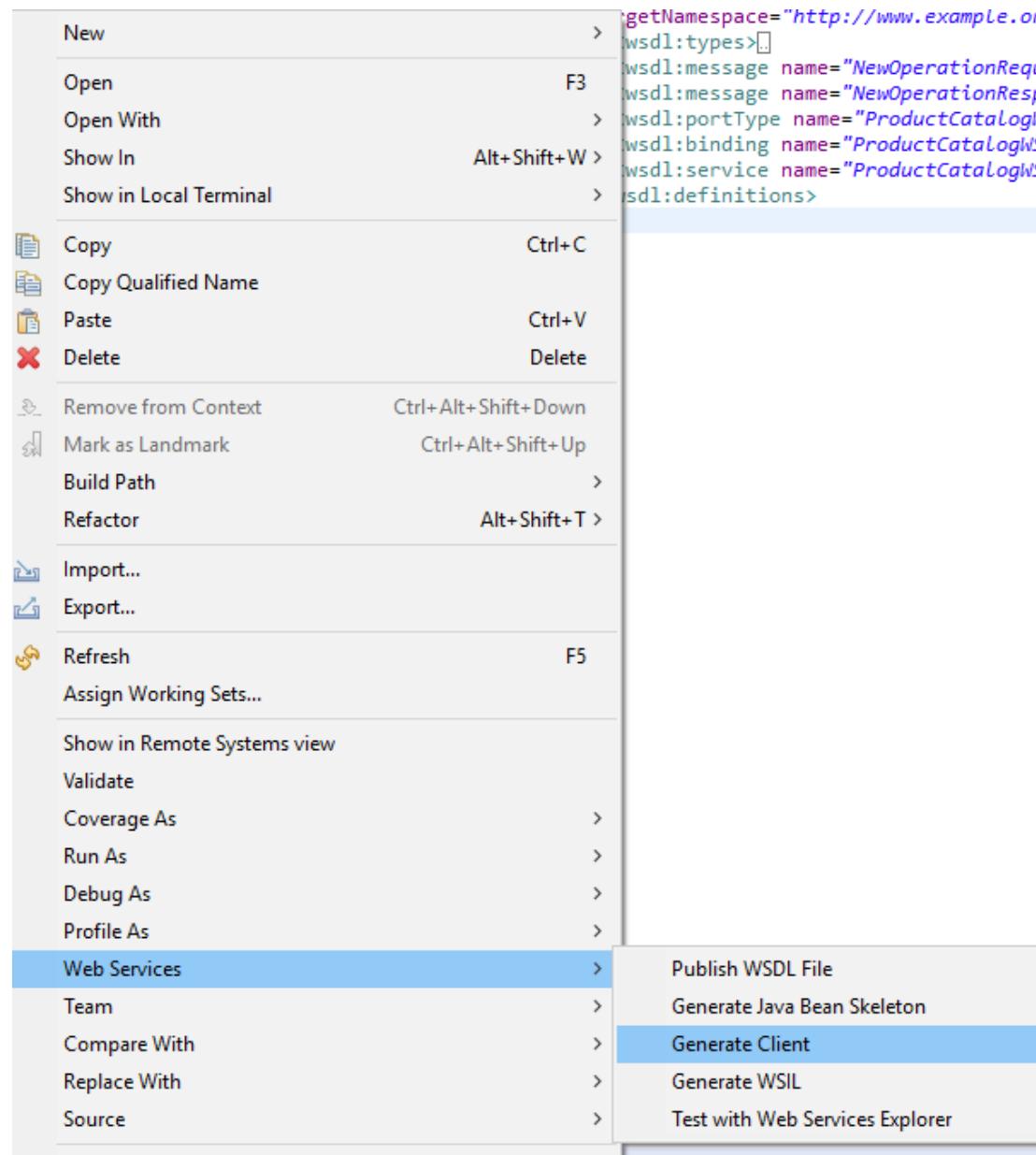
- generated wsdl file has a skeleton with the following information
 - wsdl:types
 - Wsdl:message
 - Wsdl:portType
 - Wsdl:binding
 - Wsdl:service

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.example.org/ProductCatalogWSDL/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ProductCatalogWSDL"
targetNamespace="http://www.example.org/ProductCatalogWSDL">
    <wsdl:types>...
    <wsdl:message name="NewOperationRequest">..
    <wsdl:message name="NewOperationResponse">..
    <wsdl:portType name="ProductCatalogWSDL">..
    <wsdl:binding name="ProductCatalogWSDLSOAP" type="tns:ProductCatalogWSDL">..
    <wsdl:service name="ProductCatalogWSDL">..
</wsdl:definitions>
```



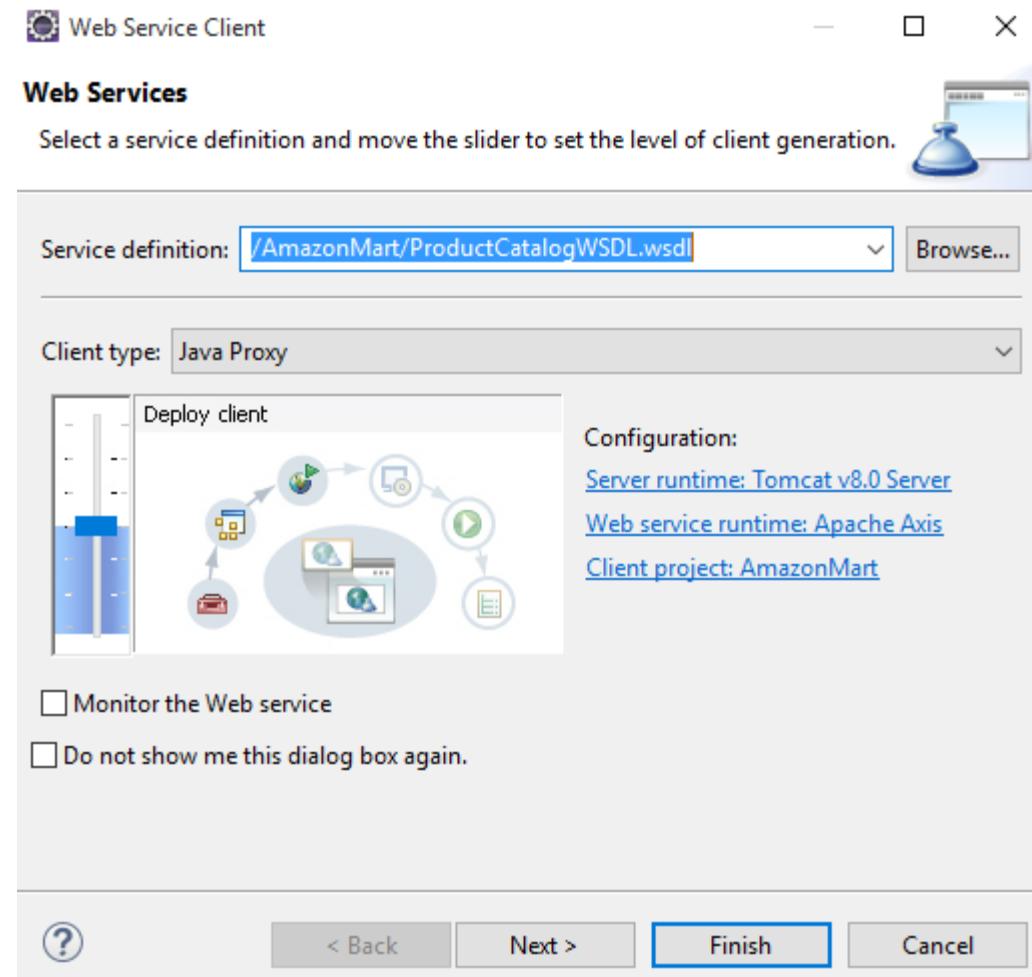
Step 7

- Generate Web Service Client from ProductCatalogWSDL.wsdl

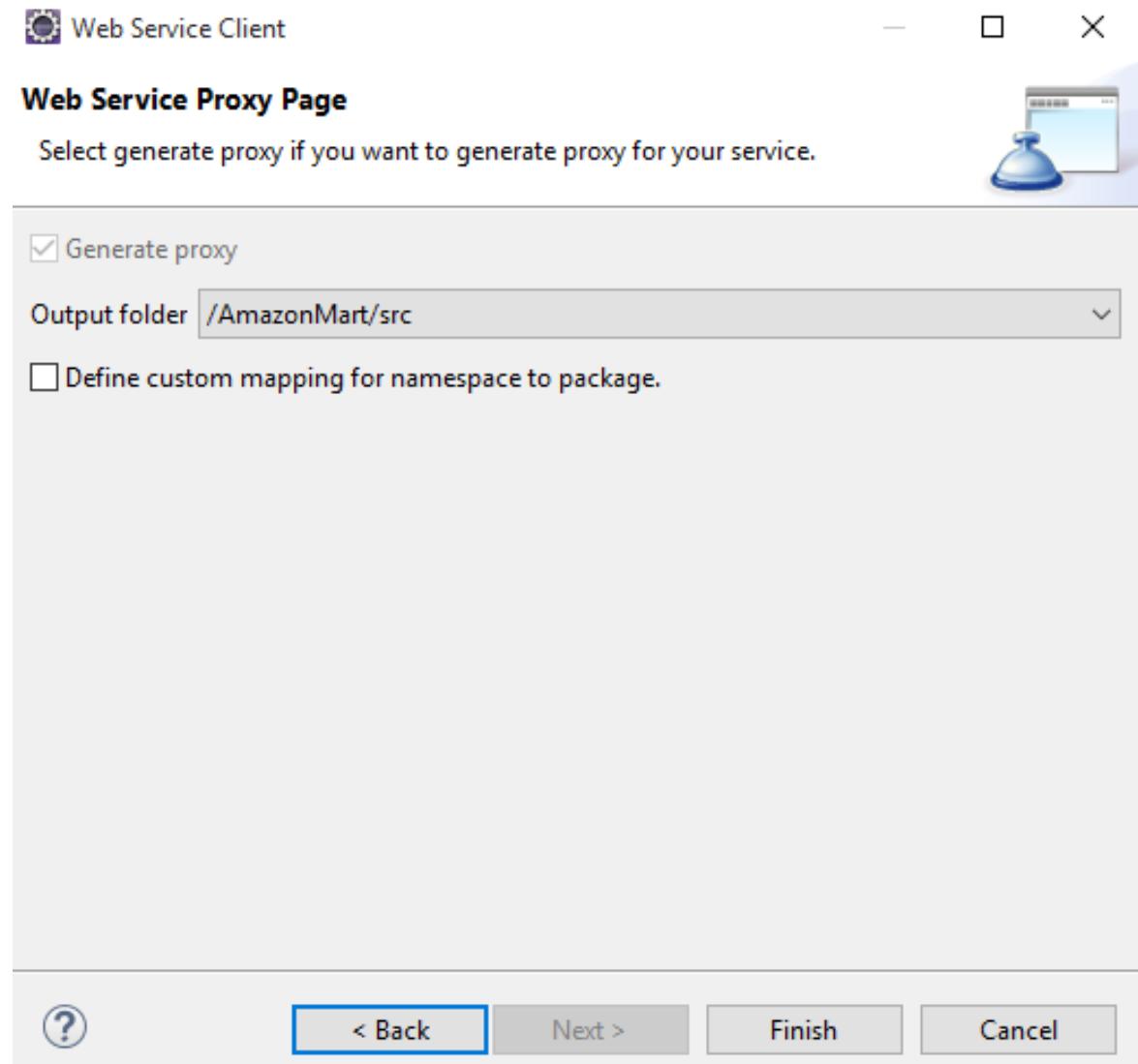


Step 7

- generate a web service client using the wsdl service definition.

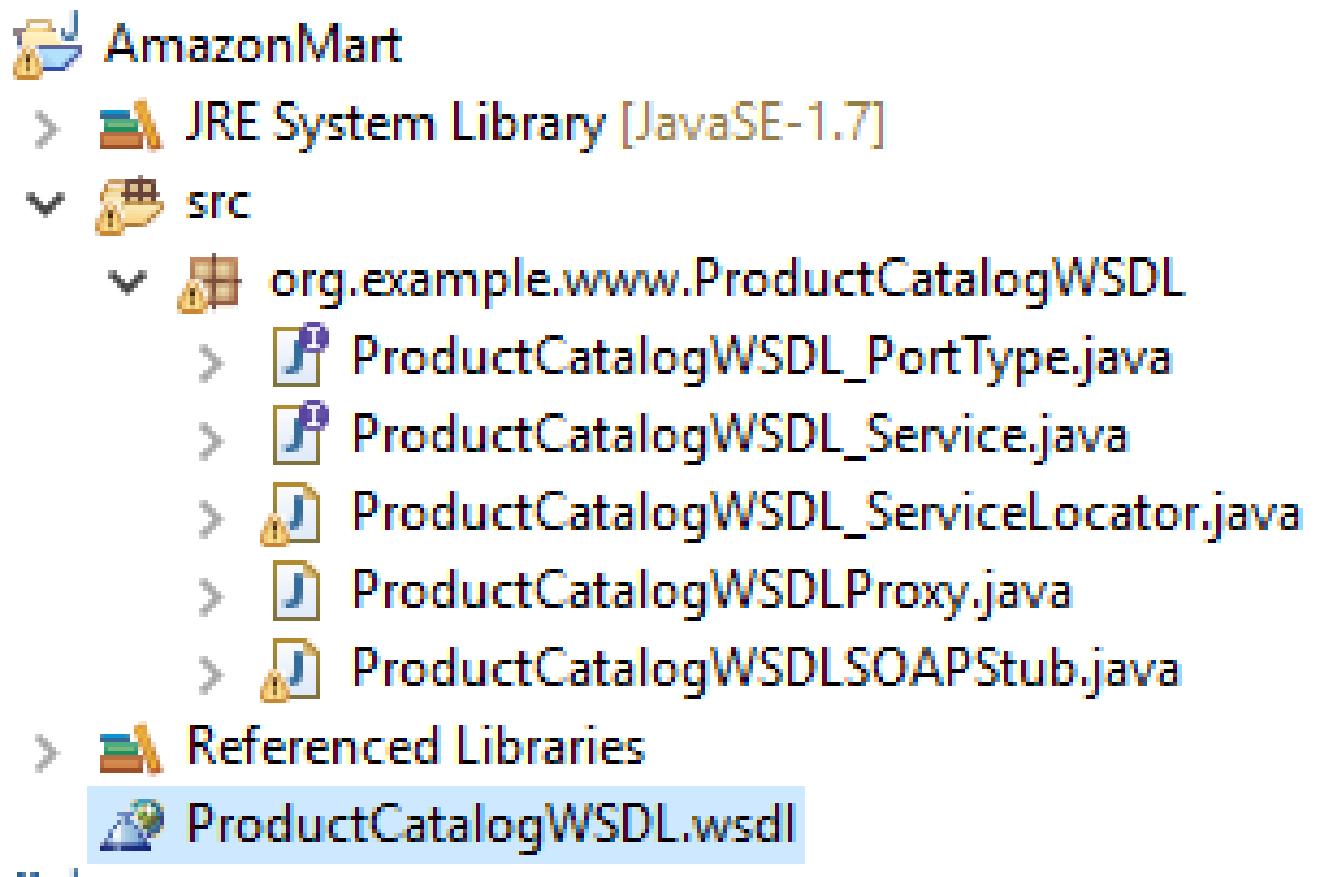


Step 8



Step 9

- generates corresponding java files for wsclient





SYED AWASE

VII: JAVA WEB SERVICES: JAX-WS

SOAP Web Service Design

Top Down or WSDL First or Contract First

1.Create WSDL file

2.Generate java stubs using tools like wsdl2java

3.Implement the web services end point

Bottom Up or Code First

1

- Write Java Code and annotate

2

- Generate the WSDL from the code using java2wsdl

3

- Implement the web services



SOAP Web Service Design

Top Down or WSDL First or Contract First

- Advantages
 - Contract with the consumer signed off
 - Better interoperability
 - Easy to read and understand the service contract
 - Faster integration

Bottom Up or Code First

- Advantages
 - Integrate legacy applications easily



XSD

- It defines a schema which is a definition of how an XML document can be structured.
- It is used to validate the structure and rules laid out in the schema for a given XML document.
- It describes the static structure of the complex data types being exchanged by those service methods. It describes, the types, their fields, any restriction on those fields and so forth.
- It's a description of data types and thus static properties of the service.

WSDL

VS

- It is an XML document that describes a web service. It shows which operations are available and how the data should be structured to send to those operations.
- They have an associated XSD that shows what is valid to put in a WSDL document.
- It's a description of the behaviour of the service and it's core functionality.
- WSDL has to adhere to a XSD
- WSDL is to describe the webservice location and operations.

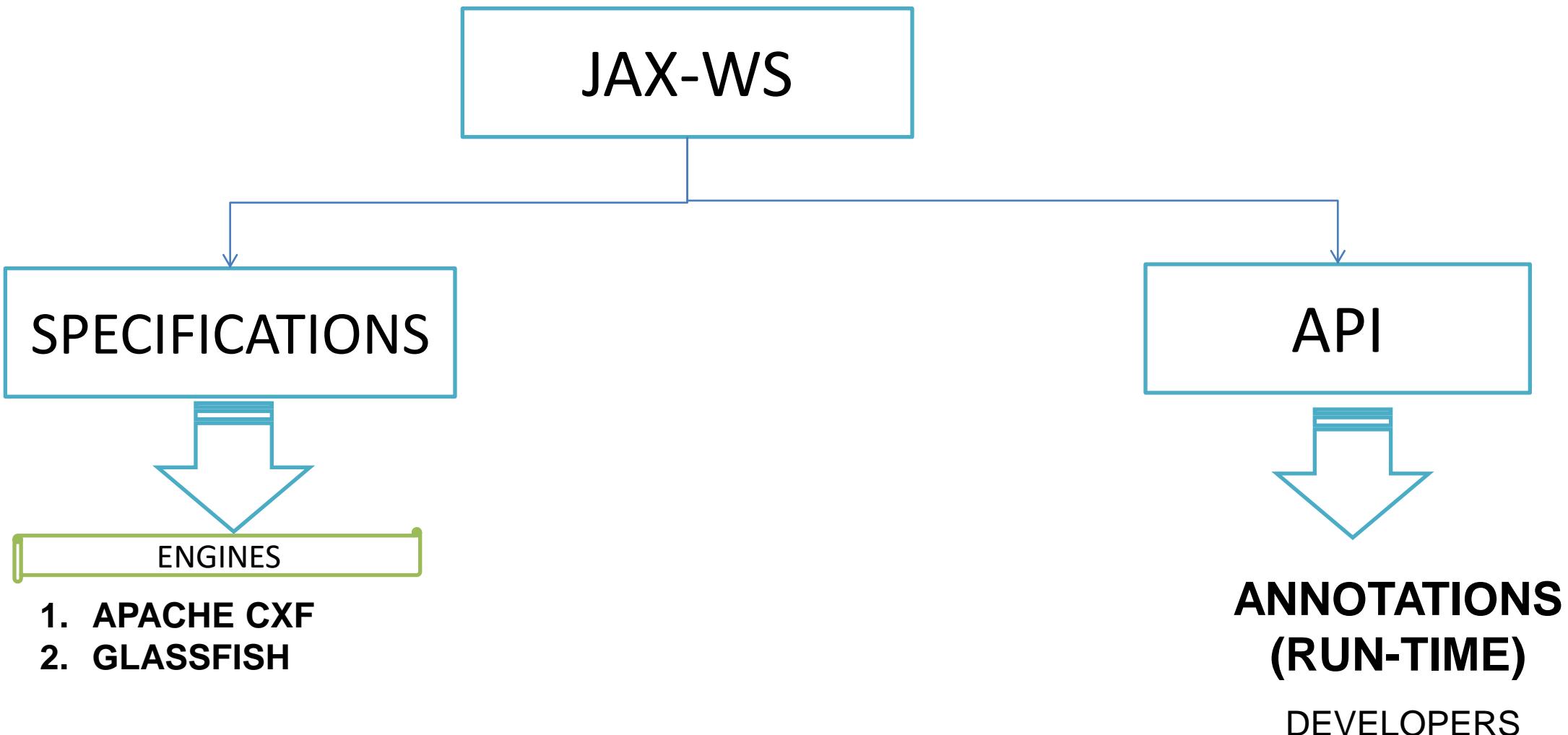


WHICH APPROACH TO CHOOSE?

- Use Contract first approach/WSDL Contract approach/ Top Down approach as much as possible as it gives the WSDL contract up front, which is readable and easy to understand.
- Use Bottom Up/Code First Approach **ONLY** for exposing legacy applications as web services



JAX-WS



JAX-WS:CORE ANNOTATIONS

ANNOTATION	DESCRIPTION
@javax.jws.WebService	Public class OrderService
@javax.jws.WebMethod	@WebResult(name="order") Order getOrder@WebParam(name="orderId) Long orderId
@javax.xml.ws.WebFault	Custom Exceptions => MyException extends Exception



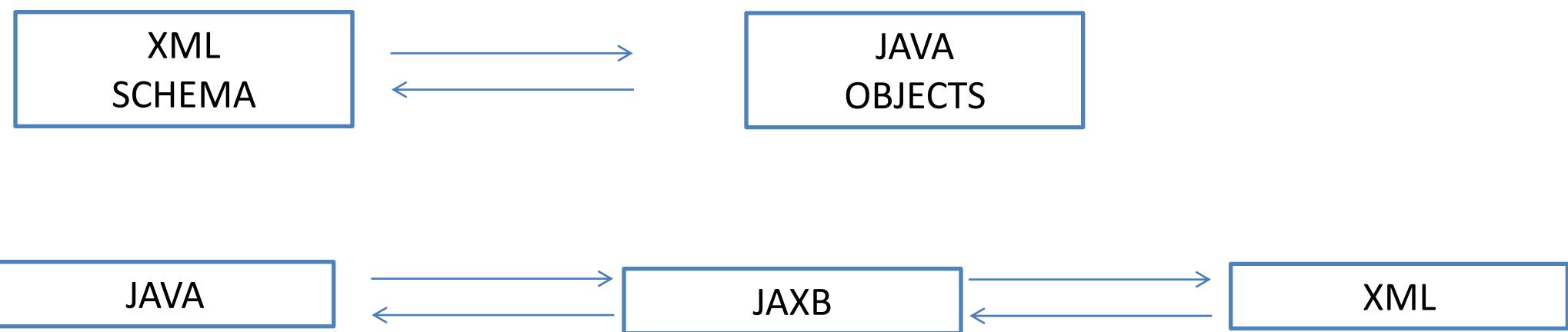
JAX-WS: SOAP BINDINGS

ANNOTATION	DESCRIPTION
@javax.jws.soap.SOAPBinding	Default: document/literal to validate the entire soap message
@SOAPBinding(style=Style.RPC,use=Use.LITERAL)	Public interface OrderService
@javax.xml.ws.RequestWrapper	Map the incoming SOAP message to java objects in a custom manner
@javax.xml.ws.ResponseWrapper	Map the incoming SOAP message to java objects in a custom manner



Java Architecture for XML Binding (JAXB)

- It provides an easy way to map java classes and XML schema, hiding the complexity of XML programming, instead of using DOM/SAX API for interacting with XML.



JAXB TOOLS

XJC

- It is used to generate java classes from an XML Schema
- Used to generate WSDL first contract internally
- @ COMPILE TIME/BUILD TIME



SCHEMAGEN

- Generates an XML schema from java classes.
- @ COMPILE TIME/BUILD TIME
- Used for code first approach/bottom up approach



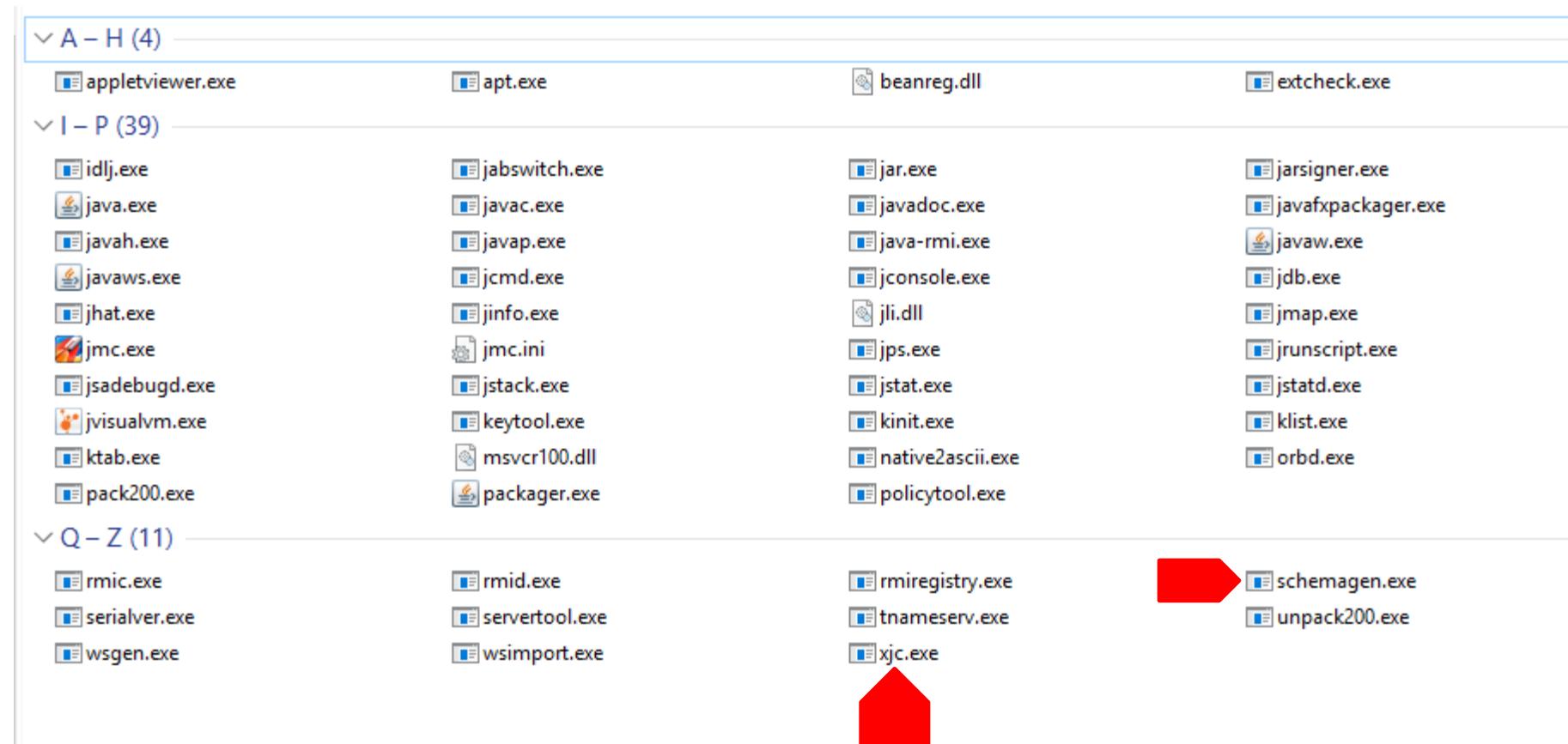
RUNTIME API

- CONVERSION OF JAVA OBJECTS TO XML OR VICE VERSA (MARSHALLING/UNMARSHAL LING)
- Marshalling class
- UnMarshalling class
- A set of Annotation



JAXB Tools packaged with JDK (from v1.6)

Local Disk (C:) > Program Files (x86) > Java > jdk1.7.0_71 > bin





SYED AWASE

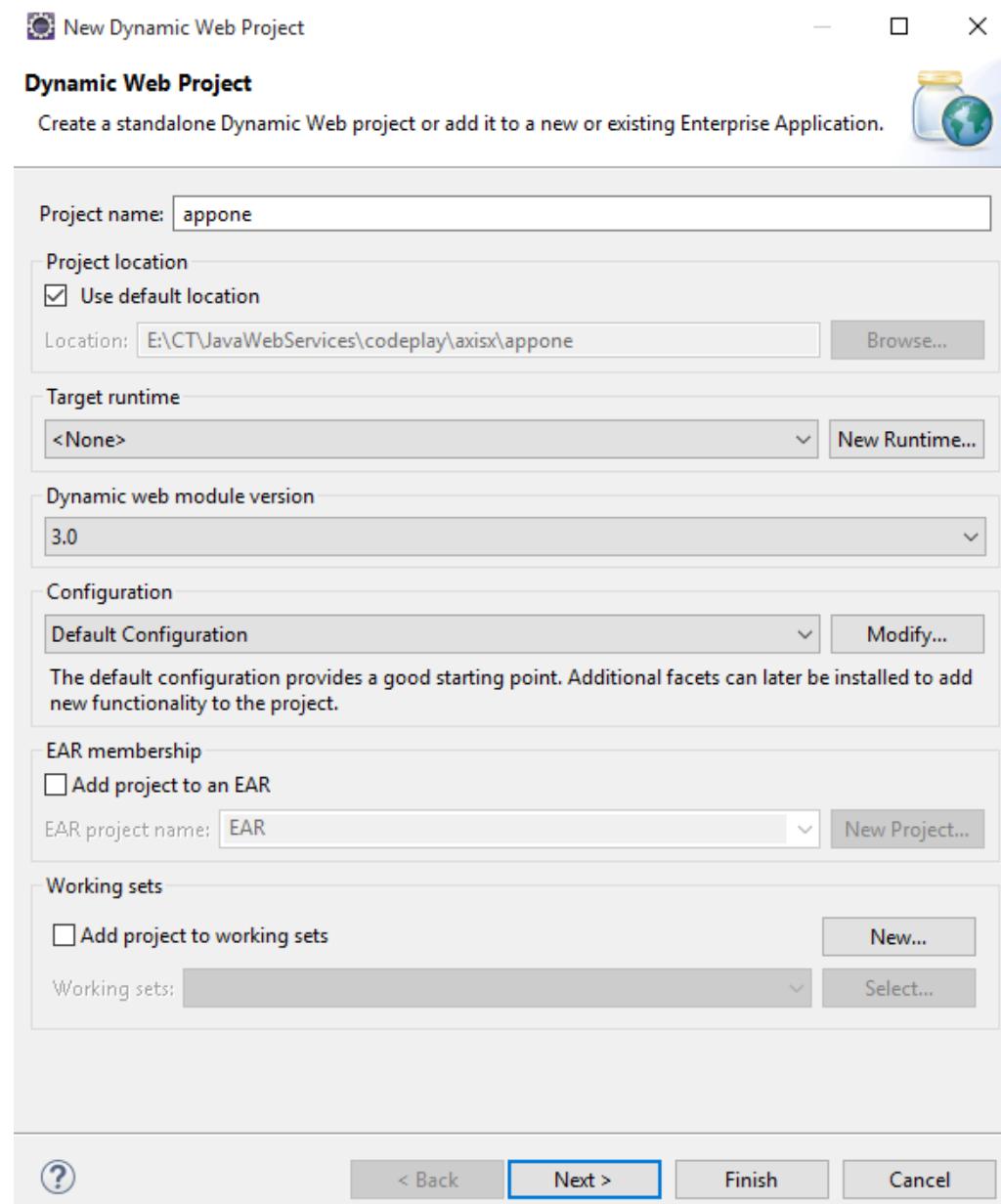
WEB SERVICES WITH AXIS

JERSEY

IX(I): PLAY BOOK – BASIC WEB SERVICE WITH APACHE AXIS 1

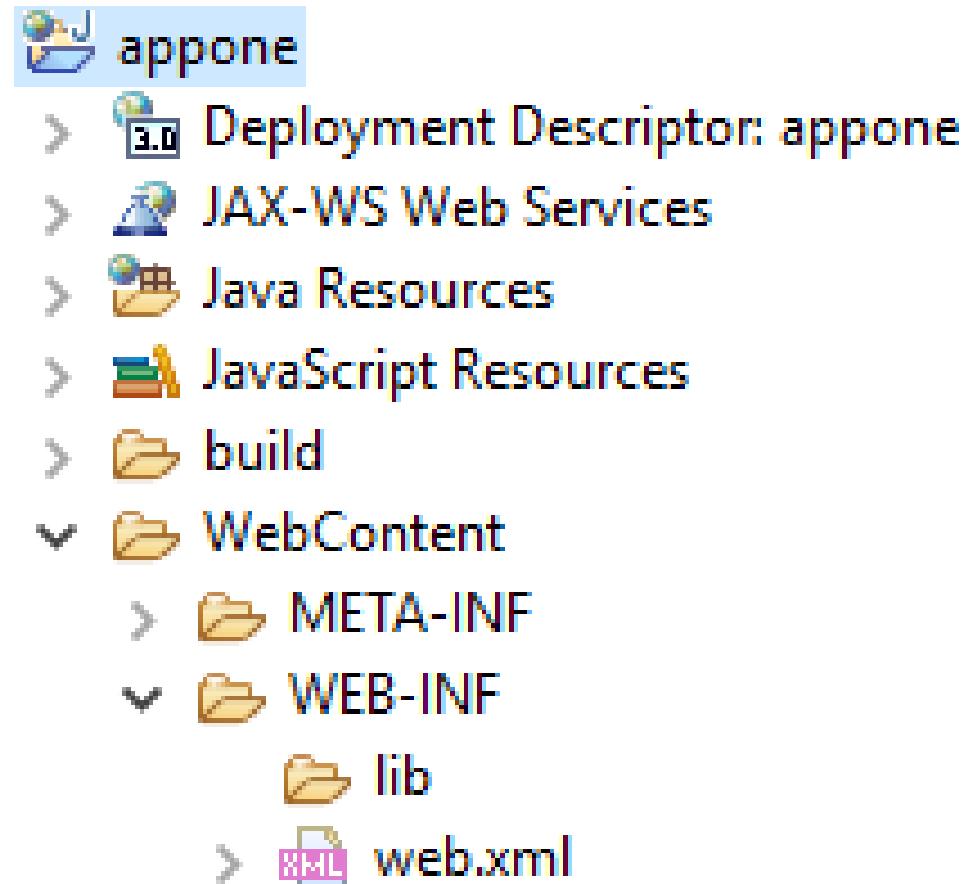
Step 1

- create a dynamic web project



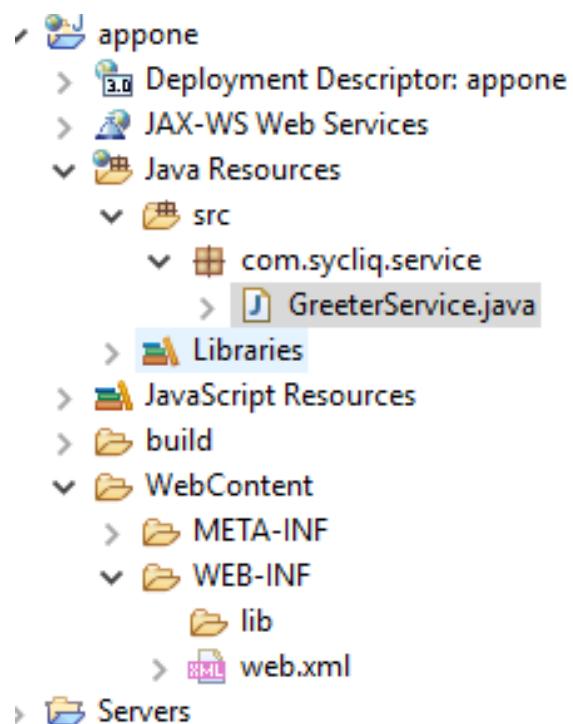
Step 2

- generated dynamic web project structure with **web.xml** configuration



Step 3

- create **GreeterService** in the package **com.syqliq.service**

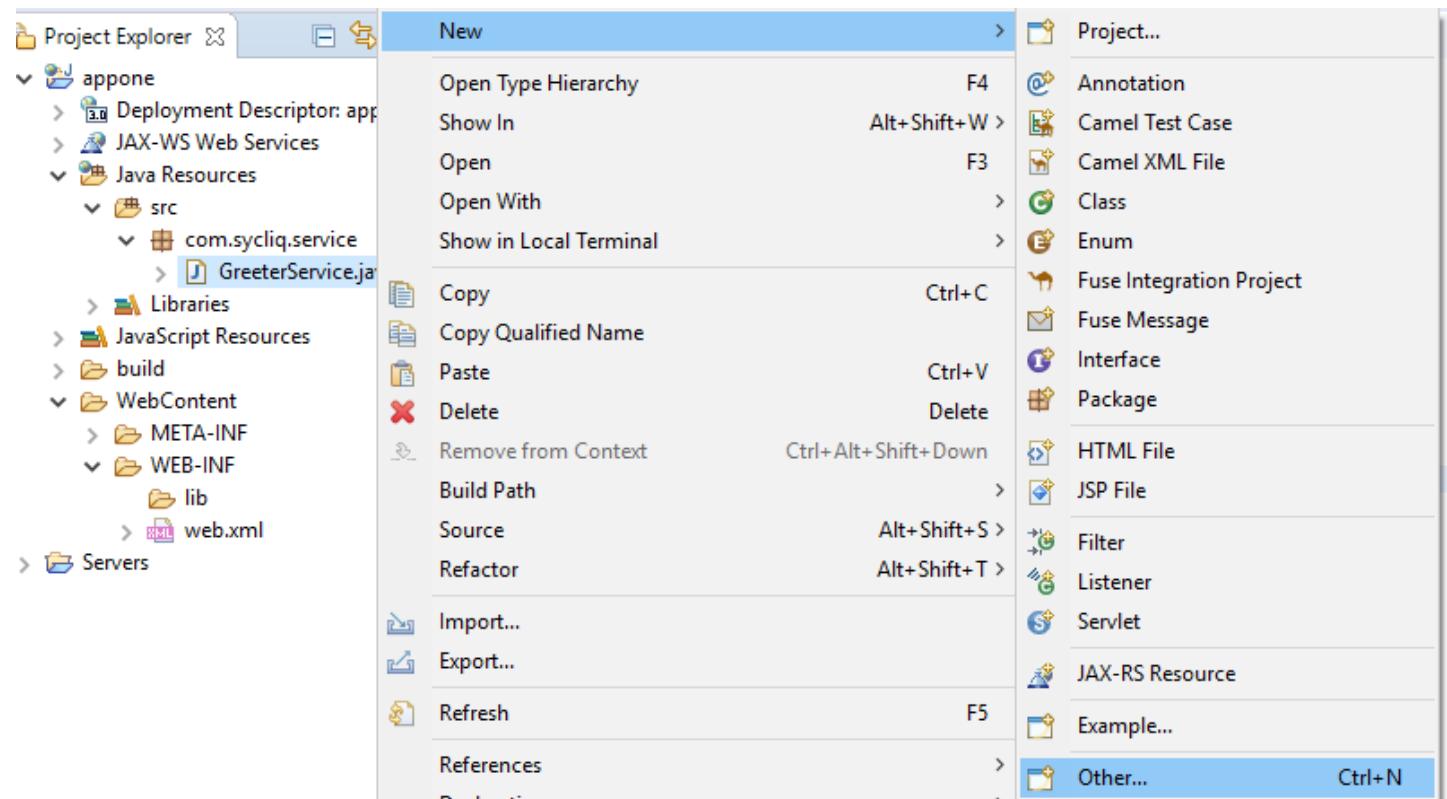


```
1 package com.syqliq.service;
2
3 public class GreeterService {
4
5     public String greetInArabic() {
6         return "Kaif Al Haal?";
7     }
8
9
10    public String greetInEnglish() {
11        return "How are you?";
12    }
13
14
15    public String greetInHindi() {
16        return "Kya Haal hain?";
17    }
18
19    public String greetInGerman() {
20        return "Wie gehts?";
21    }
22
23 }
```



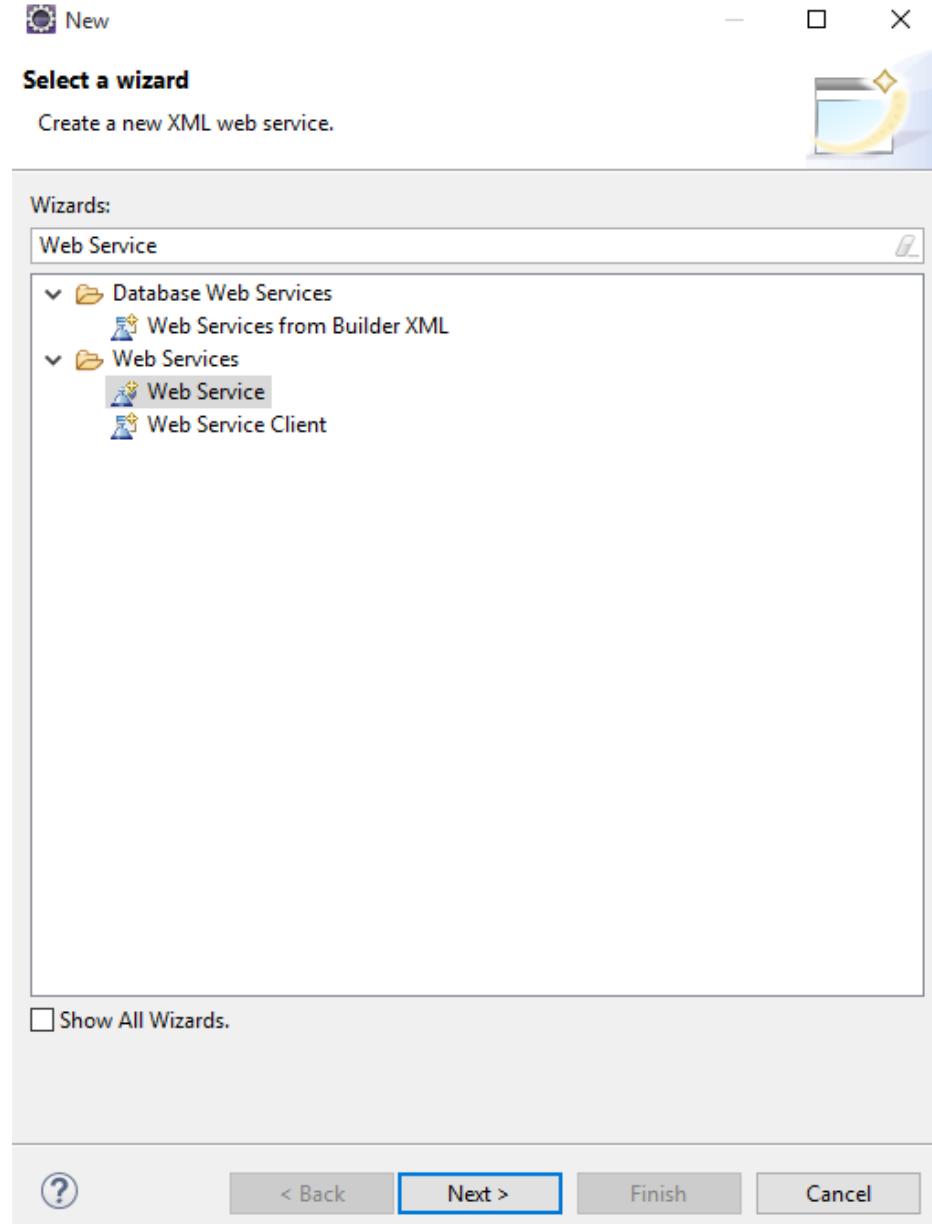
Step 4

- right click on the **GreeterService** and select other to create a new Web Service

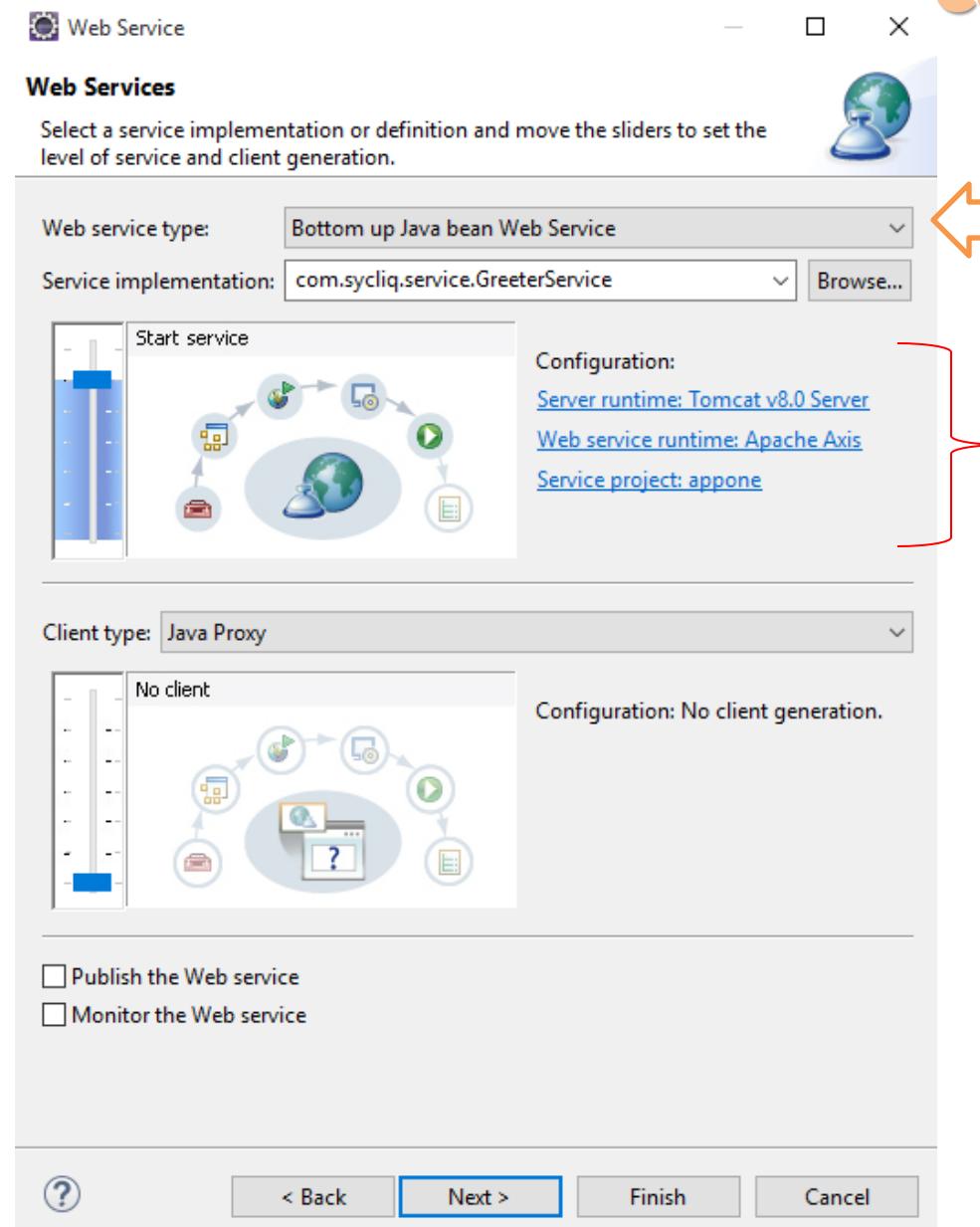


Step 5

- create a **web service** from the **GreeterService**

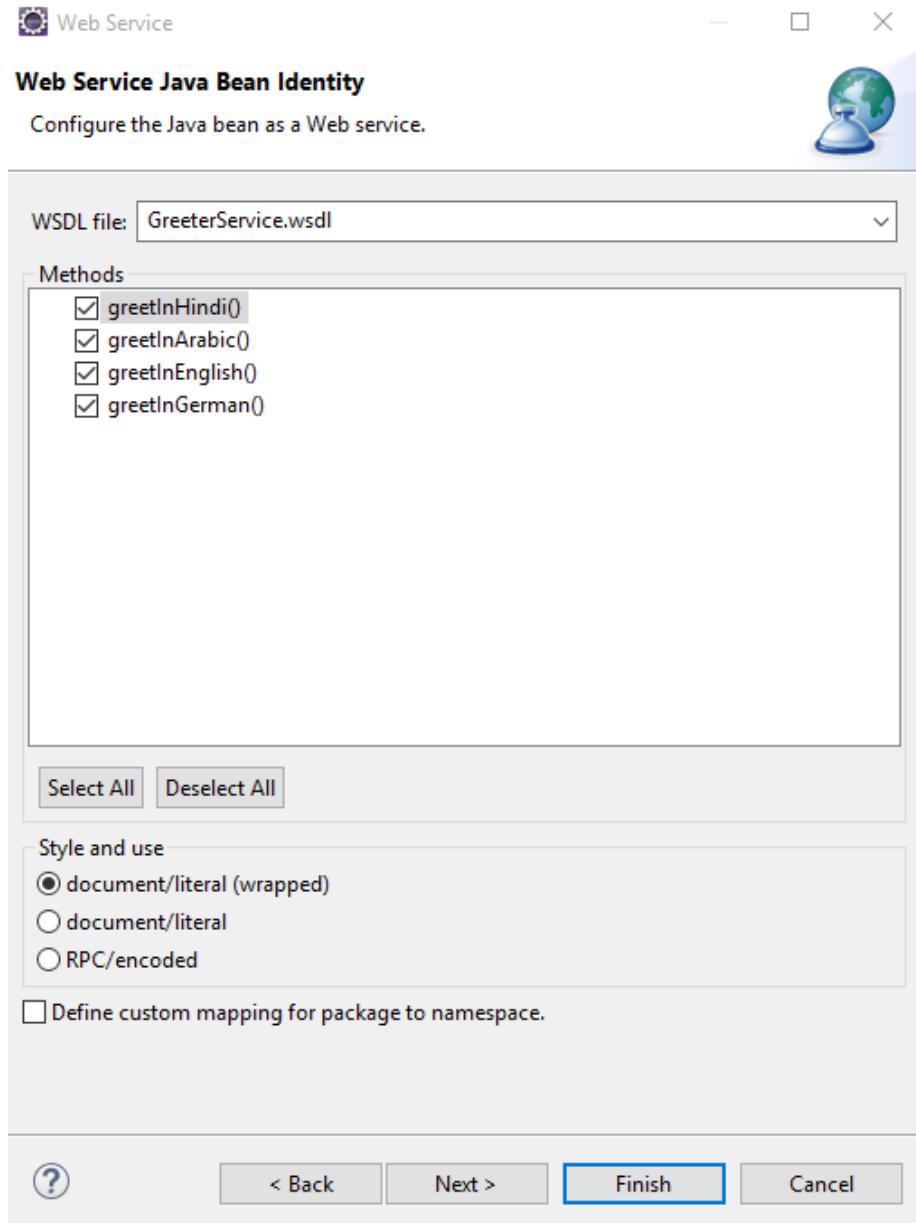


Step 6



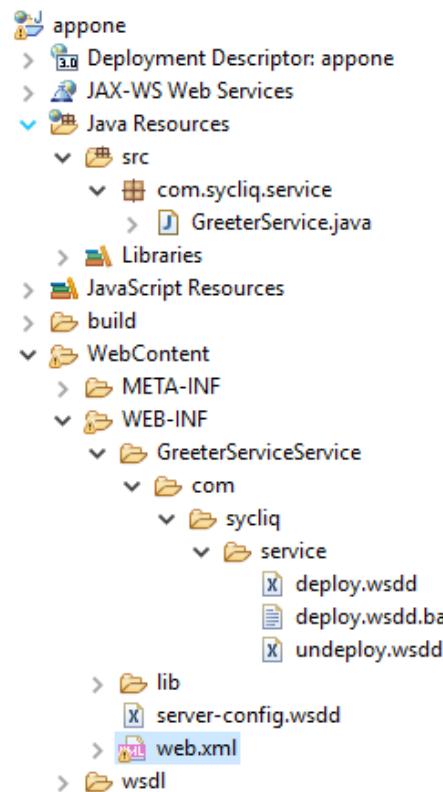
Bottom Up Approach

Step 7



Step 8

- generates wsdl and service web service for GreeterService



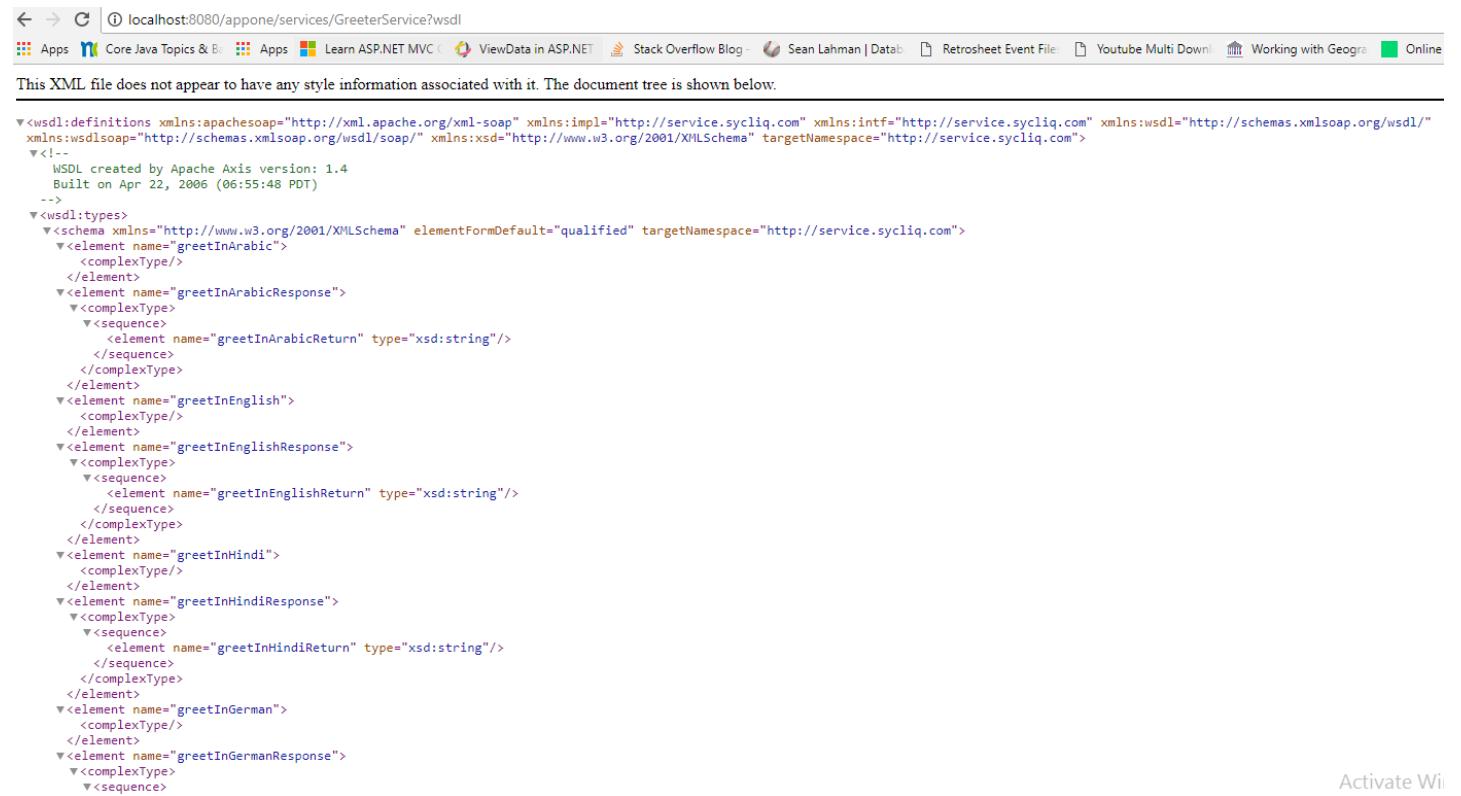
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
    <display-name>appone</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <display-name>Apache-Axis Servlet</display-name>
        <servlet-name>AxisServlet</servlet-name>
        <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/servlet/AxisServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>*.jws</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
    <servlet>
        <display-name>Axis Admin Servlet</display-name>
        <servlet-name>AdminServlet</servlet-name>
        <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
        <load-on-startup>100</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>AdminServlet</servlet-name>
        <url-pattern>/servlet/AdminServlet</url-pattern>
    </servlet-mapping>
</web-app>
```



Step 9

- view the wsdl file in the browser

<http://localhost:8080/appone/services/GreeterService?wsdl>



The screenshot shows a browser window with the URL `localhost:8080/appone/services/GreeterService?wsdl`. The page displays the XML structure of the WSDL (Web Services Description Language) document. The XML code defines various service operations and their responses, including `greetInArabic`, `greetInEnglish`, `greetInHindi`, and `greetInGerman`, each with their respective return types (`xsd:string`). The browser interface includes a navigation bar with links like Apps, Core Java Topics & Books, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, Working with Geography, and Online.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://service.sycliq.com" xmlns:intf="http://service.sycliq.com" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://service.sycliq.com">
    <!--
        WSDL created by Apache Axis version: 1.4
        Built on Apr 22, 2006 (06:55:48 PDT)
    -->
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://service.sycliq.com">
            <element name="greetInArabic">
                <complexType/>
            </element>
            <element name="greetInArabicResponse">
                <complexType>
                    <sequence>
                        <element name="greetInArabicReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="greetInEnglish">
                <complexType/>
            </element>
            <element name="greetInEnglishResponse">
                <complexType>
                    <sequence>
                        <element name="greetInEnglishReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="greetInHindi">
                <complexType/>
            </element>
            <element name="greetInHindiResponse">
                <complexType>
                    <sequence>
                        <element name="greetInHindiReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="greetInGerman">
                <complexType/>
            </element>
            <element name="greetInGermanResponse">
                <complexType>
                    <sequence>
                        <element name="greetInGermanReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </wsdl:types>
    <wsdl:message name="greetInArabicRequest">
        <wsdl:part name="parameters" type="tns:greetInArabic"/>
    </wsdl:message>
    <wsdl:message name="greetInArabicResponse">
        <wsdl:part name="parameters" type="tns:greetInArabicResponse"/>
    </wsdl:message>
    <wsdl:message name="greetInEnglishRequest">
        <wsdl:part name="parameters" type="tns:greetInEnglish"/>
    </wsdl:message>
    <wsdl:message name="greetInEnglishResponse">
        <wsdl:part name="parameters" type="tns:greetInEnglishResponse"/>
    </wsdl:message>
    <wsdl:message name="greetInHindiRequest">
        <wsdl:part name="parameters" type="tns:greetInHindi"/>
    </wsdl:message>
    <wsdl:message name="greetInHindiResponse">
        <wsdl:part name="parameters" type="tns:greetInHindiResponse"/>
    </wsdl:message>
    <wsdl:message name="greetInGermanRequest">
        <wsdl:part name="parameters" type="tns:greetInGerman"/>
    </wsdl:message>
    <wsdl:message name="greetInGermanResponse">
        <wsdl:part name="parameters" type="tns:greetInGermanResponse"/>
    </wsdl:message>
    <wsdl:operation name="greetInArabic">
        <wsdl:input message="greetInArabicRequest"/>
        <wsdl:output message="greetInArabicResponse"/>
    </wsdl:operation>
    <wsdl:operation name="greetInEnglish">
        <wsdl:input message="greetInEnglishRequest"/>
        <wsdl:output message="greetInEnglishResponse"/>
    </wsdl:operation>
    <wsdl:operation name="greetInHindi">
        <wsdl:input message="greetInHindiRequest"/>
        <wsdl:output message="greetInHindiResponse"/>
    </wsdl:operation>
    <wsdl:operation name="greetInGerman">
        <wsdl:input message="greetInGermanRequest"/>
        <wsdl:output message="greetInGermanResponse"/>
    </wsdl:operation>
</wsdl:definitions>
```

Activate Wi



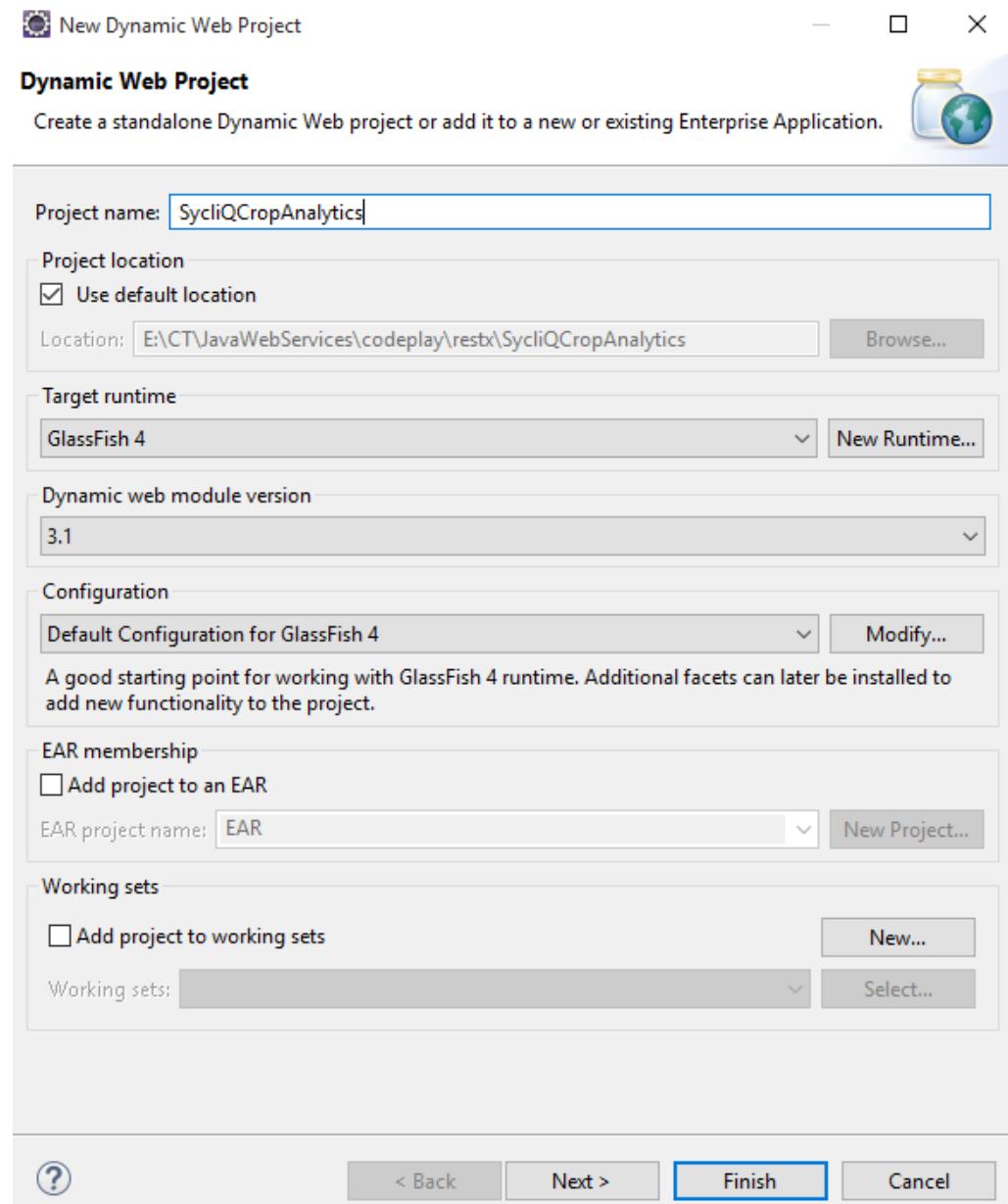


SYED AWASE

VII(A): SIMPLE WEB SERVICE USING DYNAMIC WEB PROJECT +GLASSFISH SERVER

Step 1

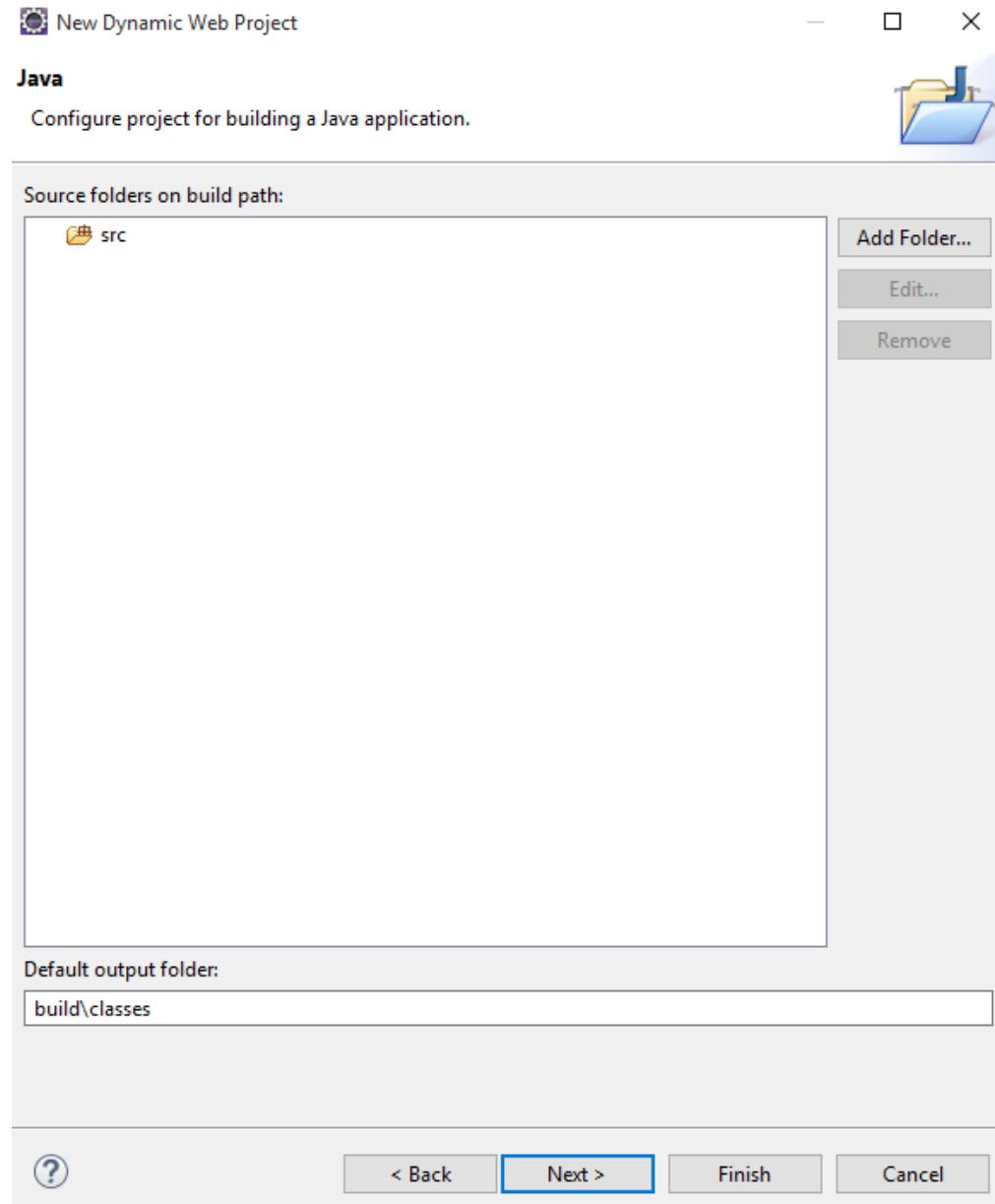
- Create a Dynamic Web Project
 - SycliQCropAnalytics





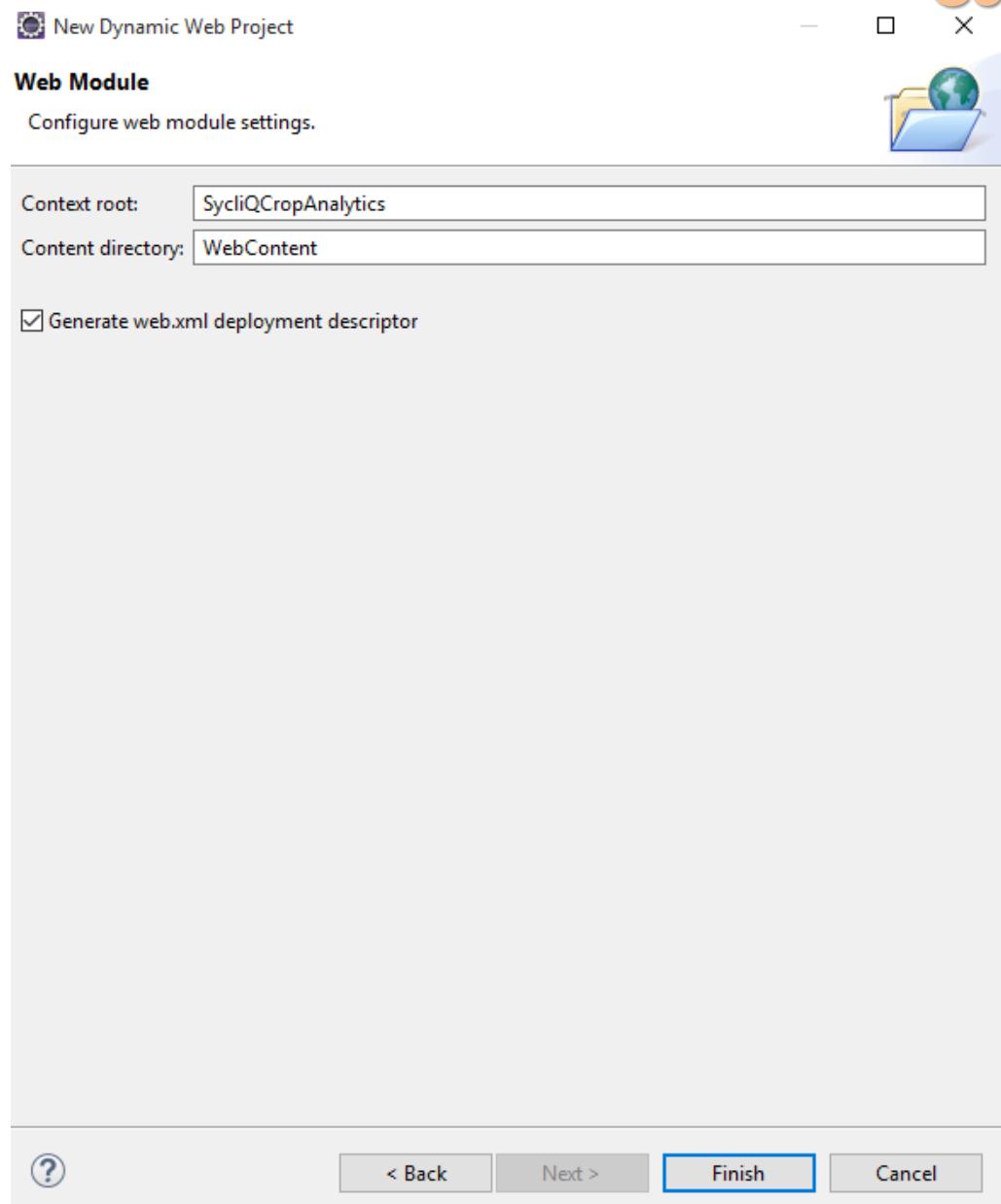
Step 2

- create project structure repository



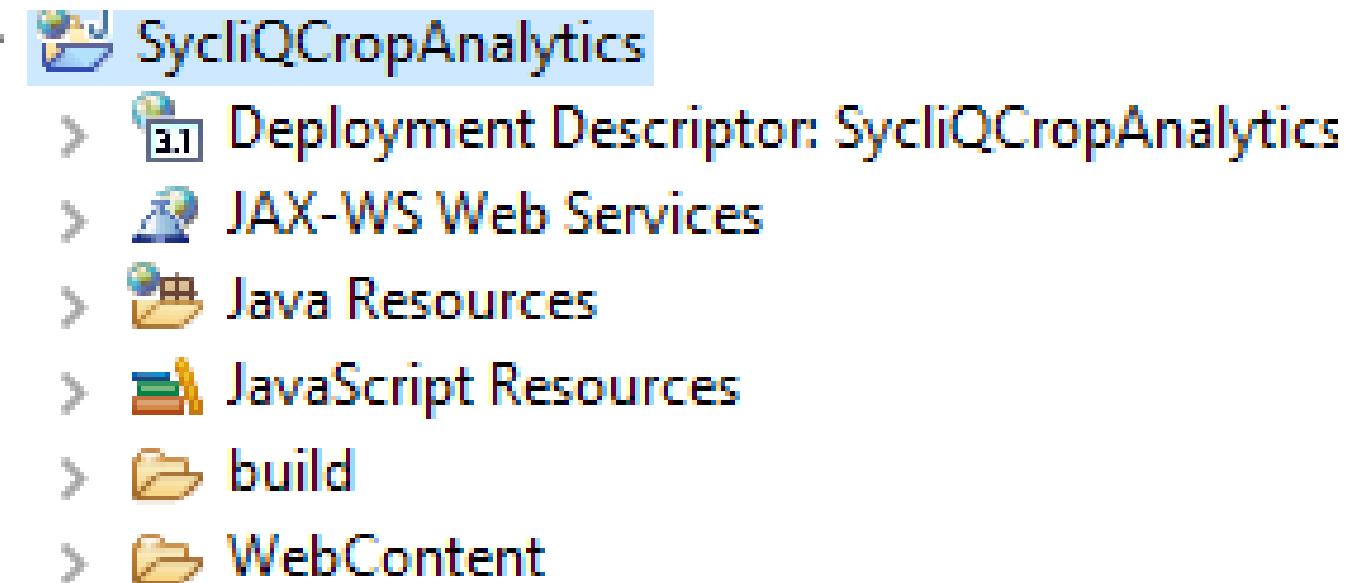
Step 3

- generate web.xml deployment descriptor



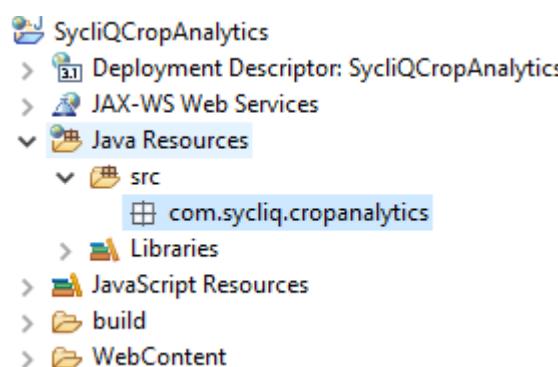
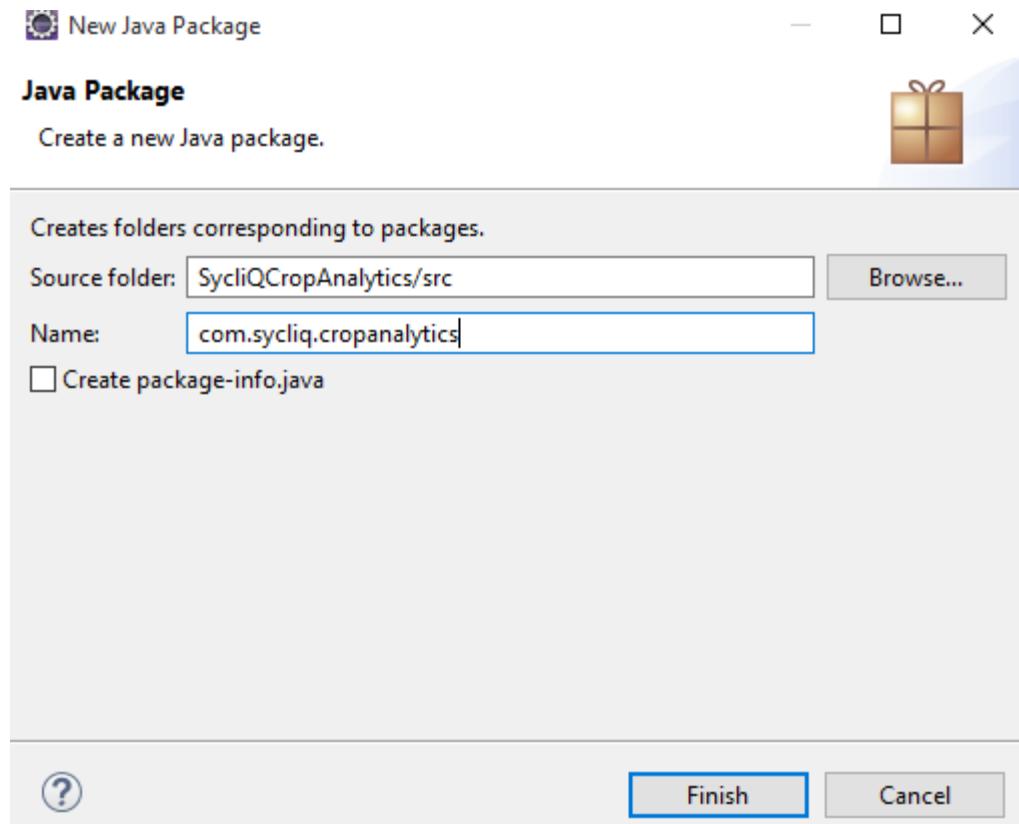
Step 4

- Dynamic web project structure generated post process.



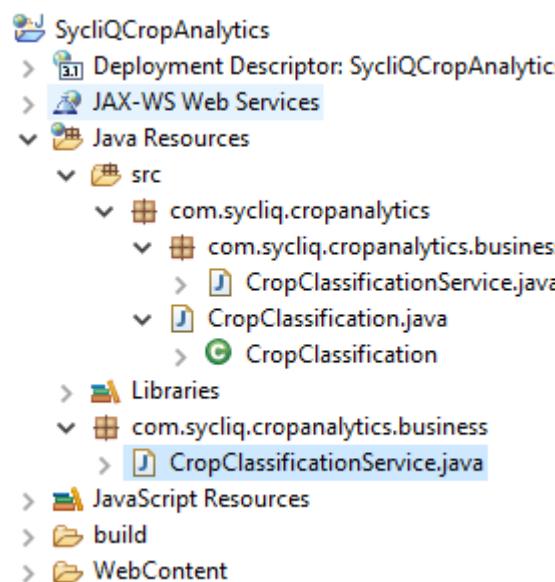
Step 5

- create a package
com.sycliq.cropanalytics



Step 6

- create a CropClassificationService business class to fetch a list of cropcategories.



```
package com.syqliq.cropanalytics.business;

import java.util.ArrayList;
import java.util.List;

public class CropClassificationService {

    public List<String> getCropCategories(){

        List<String> cropCategories = new ArrayList<>();
        cropCategories.add("cereals");
        cropCategories.add("vegetables and melons");
        cropCategories.add("fruits and nuts");
        cropCategories.add("oilseed crops");
        cropCategories.add("Root/tuber crops with high starch or insulin content");
        cropCategories.add("beverage and spice crops");
        cropCategories.add("leguminous crops");
        cropCategories.add("sugar crops");
        cropCategories.add("other crops");
        return cropCategories;
    }
}
```



Step 7

- Create a simple CropClassification Class with @WebService and @WebMethod annotations as shown in the figure
- javax.jws.WebService
- javax.jws.WebMethod

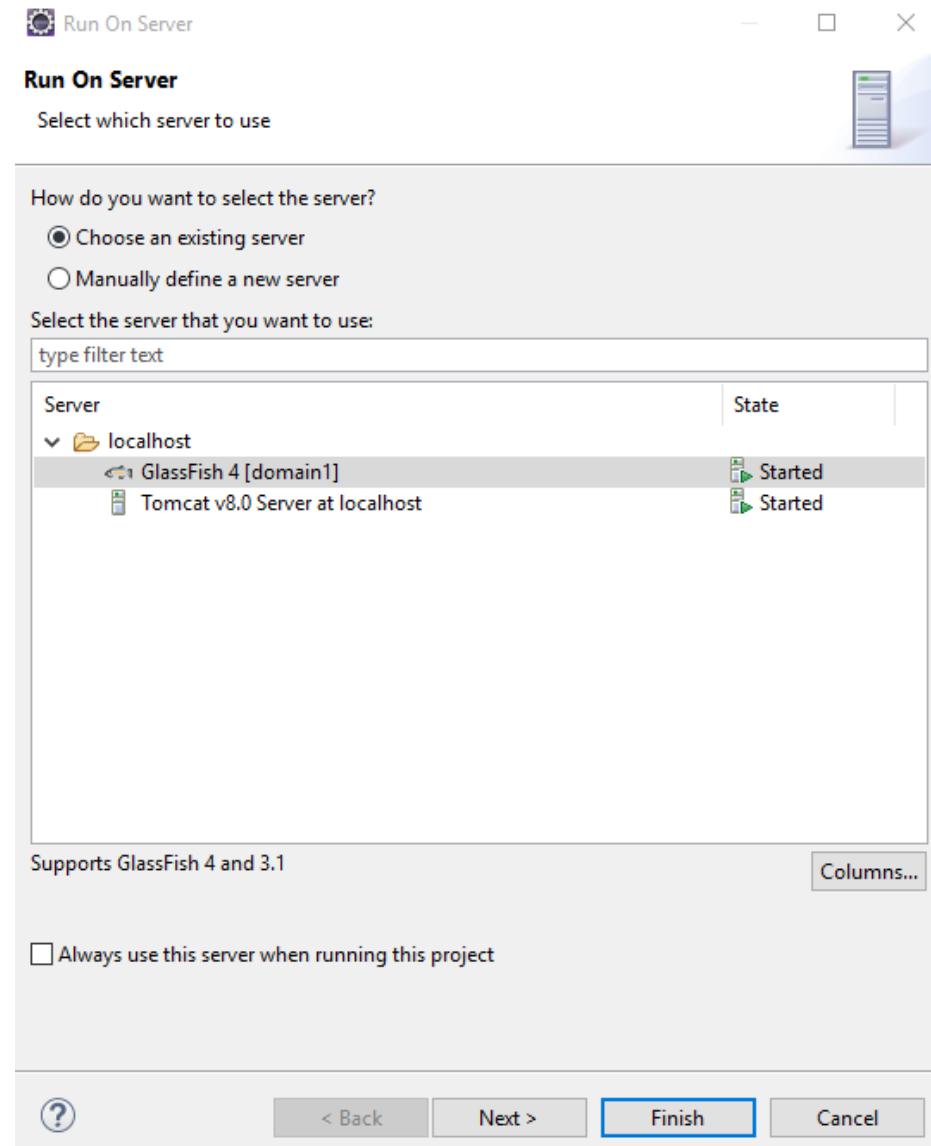
```
package com.sycliq.cropanalytics;  
import java.util.List;  
  
import javax.jws.WebMethod;  
import javax.jws.WebService;  
  
import com.sycliq.cropanalytics.business.CropClassificationService;  
  
@WebService  
public class CropClassification {  
  
    CropClassificationService cropService = new CropClassificationService();  
  
    @WebMethod  
    public List<String> getCropCategories(){  
        return cropService.getCropCategories();  
    }  
}
```





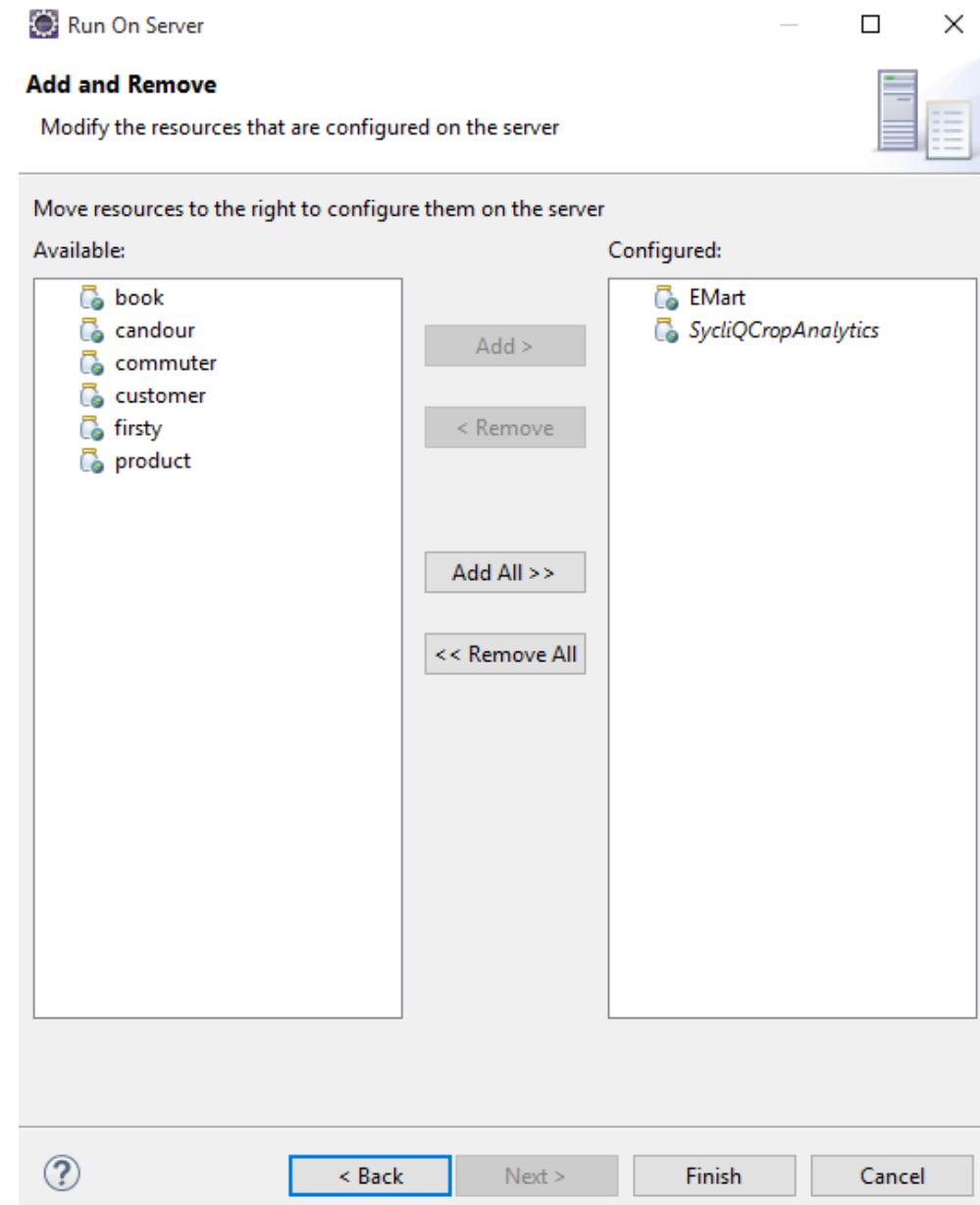
Step 8

- configure the GlassFish 4 server to render your web service



Step 9

- select the resources to render in the glassfish server.



Step 10

- the application is rendered inside glassfish applications directory

The screenshot shows the GlassFish Server Open Source Edition administration console at localhost:4848/common/index.jsf. The left sidebar has a tree view of common tasks: Domain, server (Admin Server), Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources (Concurrent Resources, Connectors, JDBC, JMS Resources, JNDI, JavaMail Sessions, Resource Adapter Configs), and Configurations (default-config, server-config, Update Tool). The Applications node is selected. The main panel is titled "Applications" and contains a table titled "Deployed Applications (2)". The table has columns: Select, Name, Deployment Order, Enabled, Engines, and Action. It lists two applications: EMart (Deployment Order 100, Enabled checked, webservices, web, Action: Launch | Redeploy | Reload) and SyclIQcropAnalytics (Deployment Order 100, Enabled checked, webservices, web, Action: Launch | Redeploy | Reload).

Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	EMart	100	<input checked="" type="checkbox"/>	webservices, web	Launch Redeploy Reload
<input type="checkbox"/>	SyclIQcropAnalytics	100	<input checked="" type="checkbox"/>	webservices, web	Launch Redeploy Reload





Step 11

- View Endpoint to render the wsdl file and the tester file to test your webservice.

The screenshot shows the GlassFish administration console at localhost:4848/common/index.jsf. The left sidebar navigation tree includes: Domain, Clusters, Standalone Instances, Nodes, Applications (with EMart and SycliQCropAnalytics selected), Resources (Concurrent Resources, Connectors, JDBC, JMS Resources, JNDI, JavaMail Sessions, Resource Adapter Configs), Configurations (default-config, server-config, Update Tool), and Help. The main content area displays the configuration for the SycliQCropAnalytics application. The application's name is SycliQCropAnalytics, status is Enabled, and it is associated with the 'server' virtual server. The context root is set to /SycliQCropAnalytics. Implicit CDI is enabled. The deployment order is 100. A table titled 'Modules and Components (4)' lists the components: SycliQCropAnalytics (web, webservices engine, default component, Servlet type), SycliQCropAnalytics (jsp component, Servlet type), SycliQCropAnalytics (CropClassification component, Servlet type), and SycliQCropAnalytics (Launch action).

Module Name	Engines	Component Name	Type	Action
SycliQCropAnalytics	[web, webservices]	-----	-----	Launch
SycliQCropAnalytics		default	Servlet	
SycliQCropAnalytics		jsp	Servlet	
SycliQCropAnalytics		CropClassification	Servlet	View Endpoint

Step 12

- Web Application links generated for SycliQCropAnalytics => CropClassification

localhost:4848/common/applications/webApplicationLinks.jsf?appId=SycliQCropAnalytics&contextRoot=/SycliQCropAnalytics

Core Java Topics & Books Apps Learn ASP.NET MVC ViewData in ASP.NET Stack Overflow Blog - Sean Lahman | Database Retrospect

Web Application Links

If the server or listener is not running, the link may not work. In this event, check the status of the server instance. After launching the web application, use the browser's Back button to return.

Application Name: SycliQCropAnalytics

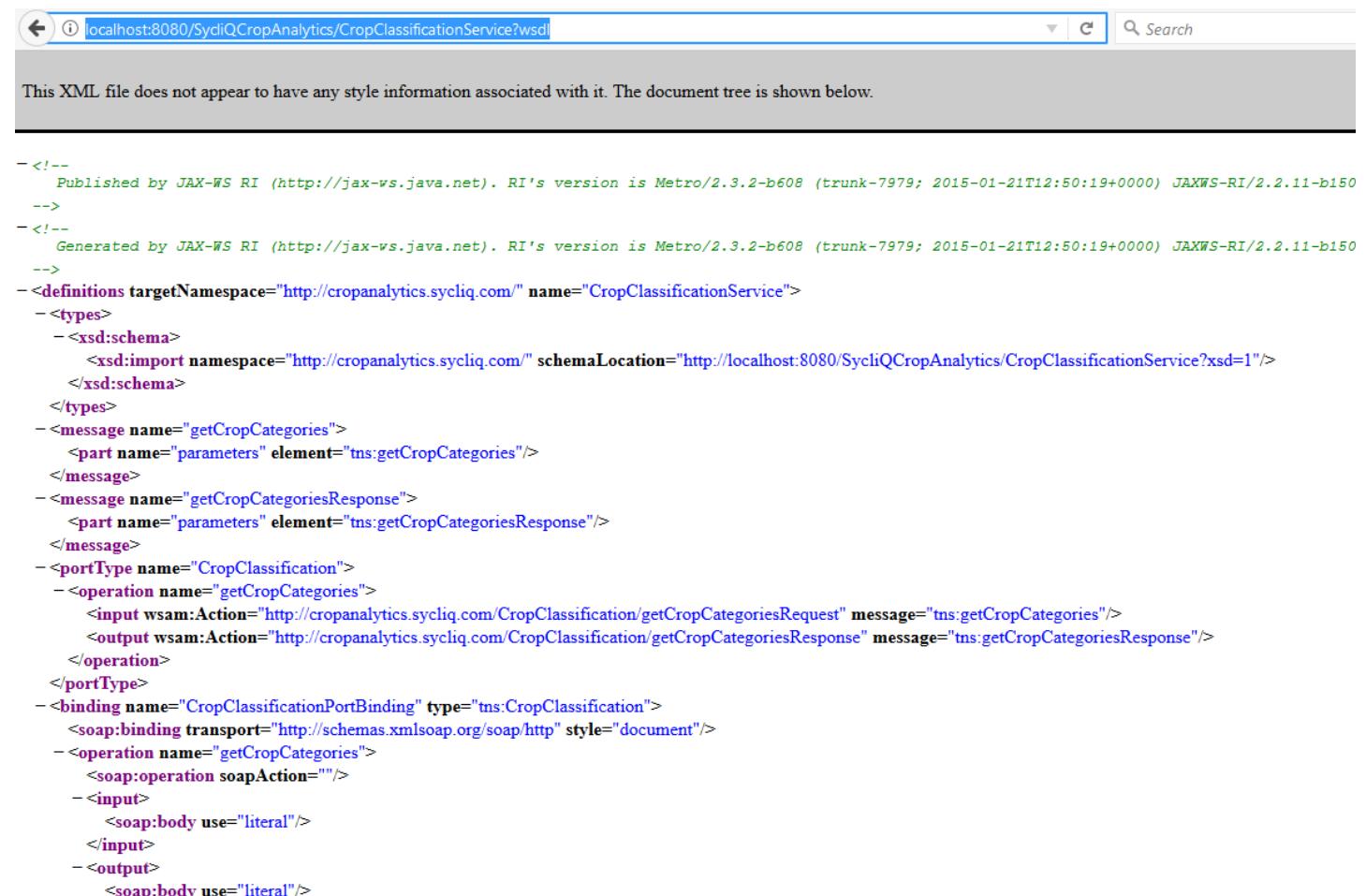
Links:

- [server] http://DESKTOP_1N4B4Y1:8080/SycliQCropAnalytics
- [server] https://DESKTOP_1N4B4Y1:8181/SycliQCropAnalytics



Step 13

- Generated wsdl file with CropClassification Service and CropClassificationPortBinding



The screenshot shows a browser window with the URL `localhost:8080/SydiQCropAnalytics/CropClassificationService?wsdl`. The page content is an XML document representing the WSDL for the CropClassificationService. The XML includes definitions for types, messages, port types, and bindings, all related to the 'getCropCategories' operation.

```
<!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150 -->
<!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150 -->
<definitions targetNamespace="http://cropanalytics.syqliq.com/" name="CropClassificationService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://cropanalytics.syqliq.com/" schemaLocation="http://localhost:8080/SydiQCropAnalytics/CropClassificationService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getCropCategories">
    <part name="parameters" element="tns:getCropCategories"/>
  </message>
  <message name="getCropCategoriesResponse">
    <part name="parameters" element="tns:getCropCategoriesResponse"/>
  </message>
  <portType name="CropClassification">
    <operation name="getCropCategories">
      <input wsam:Action="http://cropanalytics.syqliq.com/CropClassification/getCropCategoriesRequest" message="tns:getCropCategories"/>
      <output wsam:Action="http://cropanalytics.syqliq.com/CropClassification/getCropCategoriesResponse" message="tns:getCropCategoriesResponse"/>
    </operation>
  </portType>
  <binding name="CropClassificationPortBinding" type="tns:CropClassification">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getCropCategories">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
```





Step 14

- CropClassificationService Web Service Tester

localhost:8080/SycliQCropAnalytics/CropClassificationService?Tester

CropClassificationService Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

```
public abstract java.util.List com.sycliq.cropanalytics.CropClassification.getCropCategories()  
getCropCategories 0
```

Step 15

- test your web services and displays
 - SOAP Request
 - SOAP Response

The screenshot shows a web browser window with the URL `localhost:8080/SydiQCropAnalytics/CropClassificationService?Tester`. The page displays a SOAP request and its corresponding response.

Method parameter(s)

Type	Value
------	-------

Method returned

```
java.util.List : "[cereals, vegetables and melons, fruits and nuts, oilseed crops, Root/tuber crops with high starch or insulin content, beverage and spice crops, leguminous crops, sugar crops, other crops]"
```

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:getCropCategories xmlns:ns2="http://cropanalytics.syqliq.com/">
</S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:getCropCategoriesResponse xmlns:ns2="http://cropanalytics.syqliq.com/">
<return>cereals</return>
<return>vegetables and melons</return>
<return>fruits and nuts</return>
<return>oilseed crops</return>
<return>Root/tuber crops with high starch or insulin content</return>
<return>beverage and spice crops</return>
<return>leguminous crops</return>
<return>sugar crops</return>
<return>other crops</return>
</ns2:getCropCategoriesResponse>
</S:Body>
</S:Envelope>
```

Activate W
Go to Settings



Step 16

- modifying your service to accept input parameters
- Add additional methods to fetch list of crops based on categories.

```
public class CropClassificationService {  
  
    List<String> cerealslist = new ArrayList<>();  
    List<String> vmlist = new ArrayList<>();  
    List<String> fnlist = new ArrayList<>();  
    public CropClassificationService() {  
        cerealslist.add("rice");  
        cerealslist.add("dal");  
        cerealslist.add("oats");  
        cerealslist.add("wheat");  
  
        vmlist.add("bittergourd");  
        vmlist.add("ladyfinger");  
        vmlist.add("beans");  
  
        fnlist.add("papaya");  
        fnlist.add("kiwi");  
        fnlist.add("mango");  
    }  
    public List<String> getCropCategories(){  
        List<String> cropCategories = new ArrayList<>();  
        cropCategories.add("cereals");  
        cropCategories.add("vegetables and melons");  
        cropCategories.add("fruits and nuts");  
        cropCategories.add("oilseed crops");  
        cropCategories.add("Root/tuber crops with high starch or insulin content");  
        cropCategories.add("beverage and spice crops");  
        cropCategories.add("leguminous crops");  
        cropCategories.add("sugar crops");  
        cropCategories.add("other crops");  
        return cropCategories;  
    }  
    public List<String> getCrops(String CropCategory){  
        switch(CropCategory.toLowerCase()) {  
            case "cereals":  
                return cerealslist;  
            case "vegetables":  
                return vmlist;  
            case "fruits":  
                return fnlist;  
        }  
        return null;  
    }  
}
```



Step 17

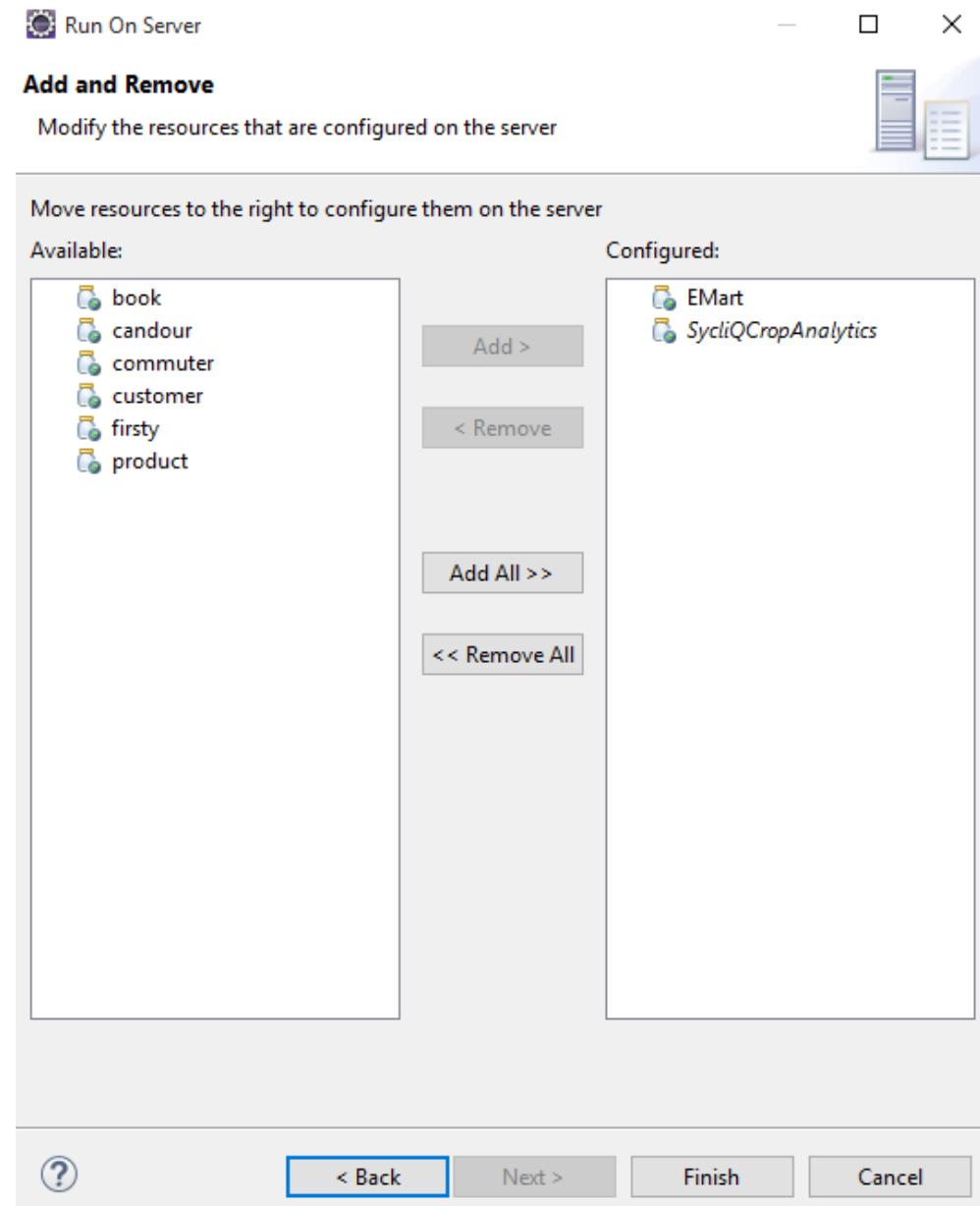
- Update the CropClassification Endpoint view to list the crops within a specific category.

```
package com.sycliq.cropanalytics;  
  
import java.util.List;  
  
import javax.jws.WebMethod;  
import javax.jws.WebService;  
  
import com.sycliq.cropanalytics.business.CropClassificationService;  
  
@WebService  
public class CropClassification {  
  
    CropClassificationService cropService = new CropClassificationService();  
  
    @WebMethod  
    public List<String> getCropCategories(){  
        return cropService.getCropCategories();  
    }  
  
    @WebMethod  
    public List<String> getCrops(String category){  
        return cropService.getCrops(category);  
    }  
}
```



Step 18

- Redeploy the application on to Glassfish application server



Step 19

- run your cropclassification service web service tester
- we see an input parameter for getCrops => “fruits”

localhost:9090/SycliQCropAnalytics/CropClassificationService?Tester | Search

CropClassificationService Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

```
public abstract java.util.List com.sycliq.cropanalytics.CropClassification.getCropCategories()  
getCropCategories 0
```

```
public abstract java.util.List com.sycliq.cropanalytics.CropClassification.getCrops(java.lang.String)  
getCrops (fruits )
```



Step 20

- Invoking the getcrops method from the web service tester displays the SOAP Request and Response Object as shown

localhost:9090/SydiQCropAnalytics/CropClassificationService?Tester

getcrops Method invocation

Method parameter(s)

Type	Value
java.lang.String	fruits

Method returned

java.util.List : "[papaya, kiwi, mango]"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:getCrops xmlns:ns2="http://cropanalytics.syqliq.com/">
<arg0>fruits</arg0>
</ns2:getCrops>
</S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:getCropsResponse xmlns:ns2="http://cropanalytics.syqliq.com/">
<return>papaya</return>
<return>kiwi</return>
<return>mango</return>
</ns2:getCropsResponse>
</S:Body>
</S:Envelope>
```

Activ...
Go to S...



Step 20

- SOAP Request to CropClassification Service

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:getCrops xmlns:ns2="http://cropanalytics.sycliq.com/">
      <arg0>fruits</arg0>
    </ns2:getCrops>
  </S:Body>
</S:Envelope>
```



Step 20

- SOAP Response Service

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
    <ns2:getCropsResponse xmlns:ns2="http://cropanalytics.sycliq.com/">
        <return>papaya</return>
        <return>kiwi</return>
        <return>mango</return>
    </ns2:getCropsResponse>
</S:Body>
</S:Envelope>
```

Activ
Go to ↗



Step 21:

- adding additional method addCrop to CropClassificationService

```
public boolean addCrop(String category, String crop) {  
    switch(category.toLowerCase()) {  
        case "cereals":  
            cerealslist.add(crop);  
            break;  
        case "vegetables":  
            vmlist.add(crop);  
            break;  
        case "fruits":  
            fnlist.add(crop);  
            break;  
        default:  
            return false;  
    }  
    return true;  
}
```



Step 22

- addProduct method to the CropClassification endpoint.

```
package com.sycliq.cropanalytics;  
  
import java.util.List;  
  
@WebService  
public class CropClassification {  
  
    CropClassificationService cropService = new CropClassificationService();  
  
    @WebMethod  
    public List<String> getCropCategories(){  
        return cropService.getCropCategories();  
    }  
  
    @WebMethod  
    public List<String> getCrops(String category){  
        return cropService.getCrops(category);  
    }  
  
    @WebMethod  
    public boolean addProduct(String category, String crop) {  
        return cropService.addCrop(category,crop);  
    }  
}
```



Step 23

- Post redeployment of the service end point

CropClassificationService Web Service Tester +
localhost:9090/SydiQCropAnalytics/CropClassificationService?Tester Search

CropClassificationService Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.util.List com.sycliq.cropanalytics.CropClassification.getProducts(java.lang.String)
 ()

public abstract java.util.List com.sycliq.cropanalytics.CropClassification.getCategory()
 ()

public abstract boolean com.sycliq.cropanalytics.CropClassification.addProduct(java.lang.String,java.lang.String)
 (,)



Step 24

- addCrop Method Invocation.

localhost:9090/SydiQCropAnalytics/CropClassificationService?Tester

addProduct Method invocation

Method parameter(s)

Type	Value
java.lang.String	fruits
java.lang.String	banana

Method returned

boolean : "true"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:addProduct xmlns:ns2="http://cropanalytics.sycliq.com/">
      <arg0>fruits</arg0>
      <arg1>banana</arg1>
    </ns2:addProduct>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:addProductResponse xmlns:ns2="http://cropanalytics.sycliq.com/">
```

Activ
Go to



Step 25

- Invoking the getcrops method invocation to fetch the updated records

localhost:9090/SycliQCropAnalytics/CropClassificationService?Tester

getcrops Method invocation

Method parameter(s)

Type	Value
java.lang.String	fruits

Method returned

java.util.List : "[papaya, kiwi, mango, banana, banana]"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:getCrops xmlns:ns2="http://cropanalytics.sycliq.com/">
      <arg0>fruits</arg0>
    </ns2:getCrops>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:getCropsResponse xmlns:ns2="http://cropanalytics.sycliq.com/">
      <return>papaya</return>
      <return>kiwi</return>
      <return>mango</return>
      <return>banana</return>
      <return>banana</return>
```





SYED AWASE

VII(B): CONTRACT FIRST (WSDL FIRST) APPROACH TO CREATE A WS ENDPOINT

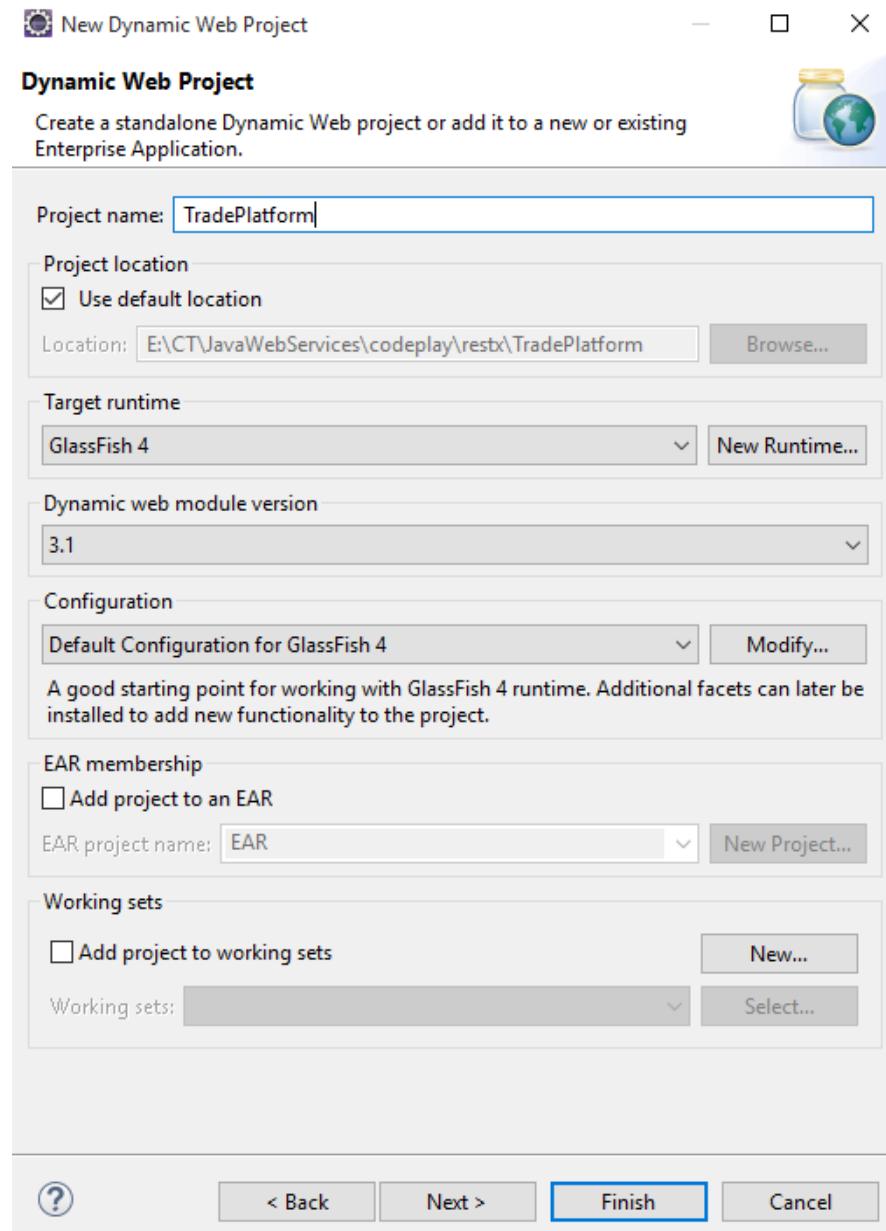
ACTIVITY OUTLINE

1. Create a WSDL file for StockData
2. Create 3 web methods to
 - a. Get the price of a stock => **getPrice()**
 - b. Update or change stock price => **updatePrice()**
 - c. Add a new stock =>**addStock()**



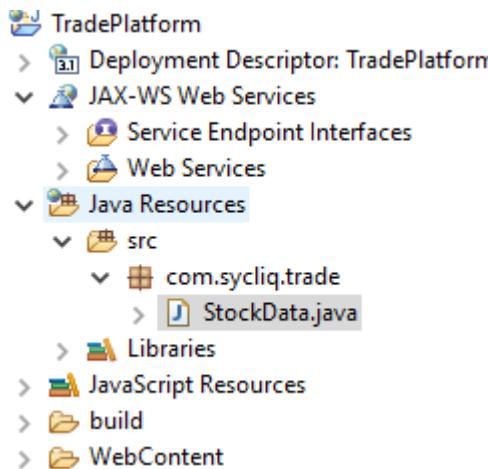
Step

- create a Dynamic Web Project with target runtime GlassFish 4



Step 2

- create a skeleton for StockData
- Publish it to glassfish server to generate wsdl file skeleton



```
package com.sycliq.trade;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService()
public class StockData {

    @WebMethod()
    public List<String> getPrice(){

        return null;
    }

    @WebMethod()
    public boolean addStock() {

        return true;
    }
}
```



Step 3

- generated wsdl file looks like this.

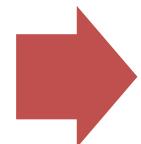
```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is  
Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12  
JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown. --><!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000)  
JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521  
svn-revision#unknown. --><definitions xmlns:wsu="  
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="  
http://www.w3.org/ns/ws-policy" xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="  
http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="  
http://trade.syqliq.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"  
targetNamespace="http://trade.syqliq.com/" name="StockDataService">  
<types>  
<message name="addStock">  
<message name="addStockResponse">  
<message name="getPrice">  
<message name="getPriceResponse">  
<portType name="StockData">  Mapped to the WebMethods  
<binding name="StockDataPortBinding" type="tns:StockData">  
<service name="StockDataService">  
<port name="StockDataPort" binding="tns:StockDataPortBinding">  
<soap:address location="https://desktop\_1n4b4y1:9191/TradePlatform/StockDataService" />  
</port>  
</service>  
</definitions>
```



Step 4

Messages

- Define the messages to map to the @WebMethods
- Request
- Response



portType

- Operation: Register various wsam Action Methods for
 - Request
 - Response



Binding

- Register binding operation for each of the WebMethods
- SOAP: body should conform to “document literal”
- If we use “encoding format” it slows the service.



Step 4(a): Messages

- create corresponding messages for input request and output response for the

WEBMETHODS

```
<message name="addStock">
<part name="parameters" element="tns:addStock"/>
</message>
<message name="addStockResponse">
<part name="parameters" element="tns:addStockResponse"/>
</message>
<message name="getPrice">
<part name="parameters" element="tns:getPrice"/>
</message>
<message name="getPriceResponse">
<part name="parameters" element="tns:getPriceResponse"/>
</message>
<message name="updatePrice">
<part name="parameters" element="tns:updatePrice"/>
</message>
<message name="updatePriceResponse">
<part name="parameters" element="tns:updatePriceResponse"/>
</message>
```



Step 4(b):portType

- configure the message by registering them to operations for input and output for each WEBMETHOD

```
<portType name="StockData">
<operation name="addStock">
<input wsam:Action="http://trade.sycliq.com/StockData/addStockRequest" message="tns:addStock"/>
<output wsam:Action="http://trade.sycliq.com/StockData/addStockResponse" message="tns:addStockResponse"/>
</operation>
<operation name="getPrice">
<input wsam:Action="http://trade.sycliq.com/StockData/getPriceRequest" message="tns:getPrice"/>
<output wsam:Action="http://trade.sycliq.com/StockData/getPriceResponse" message="tns:getPriceResponse"/>
</operation>
<operation name="updatePrice">
<input wsam:Action="http://trade.sycliq.com/StockData/updatePriceRequest" message="tns:updatePrice"/>
<output wsam:Action="http://trade.sycliq.com/StockData/updatePriceResponse" message="tns:updatePriceResponse"/>
</operation>
</portType>
```



Step 4(c): binding

```
<binding name="StockDataPortBinding" type="tns:StockData">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="addStock">
<soap:operation soapAction="" />
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="getPrice">
<soap:operation soapAction="" />
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="updatePrice">
<soap:operation soapAction="" />
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
```



addStock()

getPrice()

updatePrice()



Step 5

- Create a SOAP messages for the StockData methods
 - getPrice
 - Every node must analyze the header
 - Fail immediately if the header cannot be read
 - updatePrice
 - Only final node must analyze the header
 - Continue even if header cannot be read
 - addStock
 - Do not analyze the header



Step 6

- SOAP Message Template

```
<?xml version ="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
  <m:getPrice xmlns:m="http://stockData/StockData/getPriceRequest" env:role="
  http://www.w3.org/2003/05/soap-envelope/next" env:mustUnderstand="true">
    </m:getPrice>
  <m:updatePrice xmlns:m="http://stockData/StockData/updatePrice" env:role="http://www.w3.org/2003/05/soap-envelope/next"
  " env:mustUnderstand="false">
    </m:updatePrice>
  <m:addStock xmlns:m="http://stockData/StockData/addStock" env:role="http://www.w3.org/2003/05/soap-envelope/next" env:
  mustUnderstand="false">
    </m:addStock>
  </env:Header>
<env:Body>
</env:Body>
</env:Envelope>
```



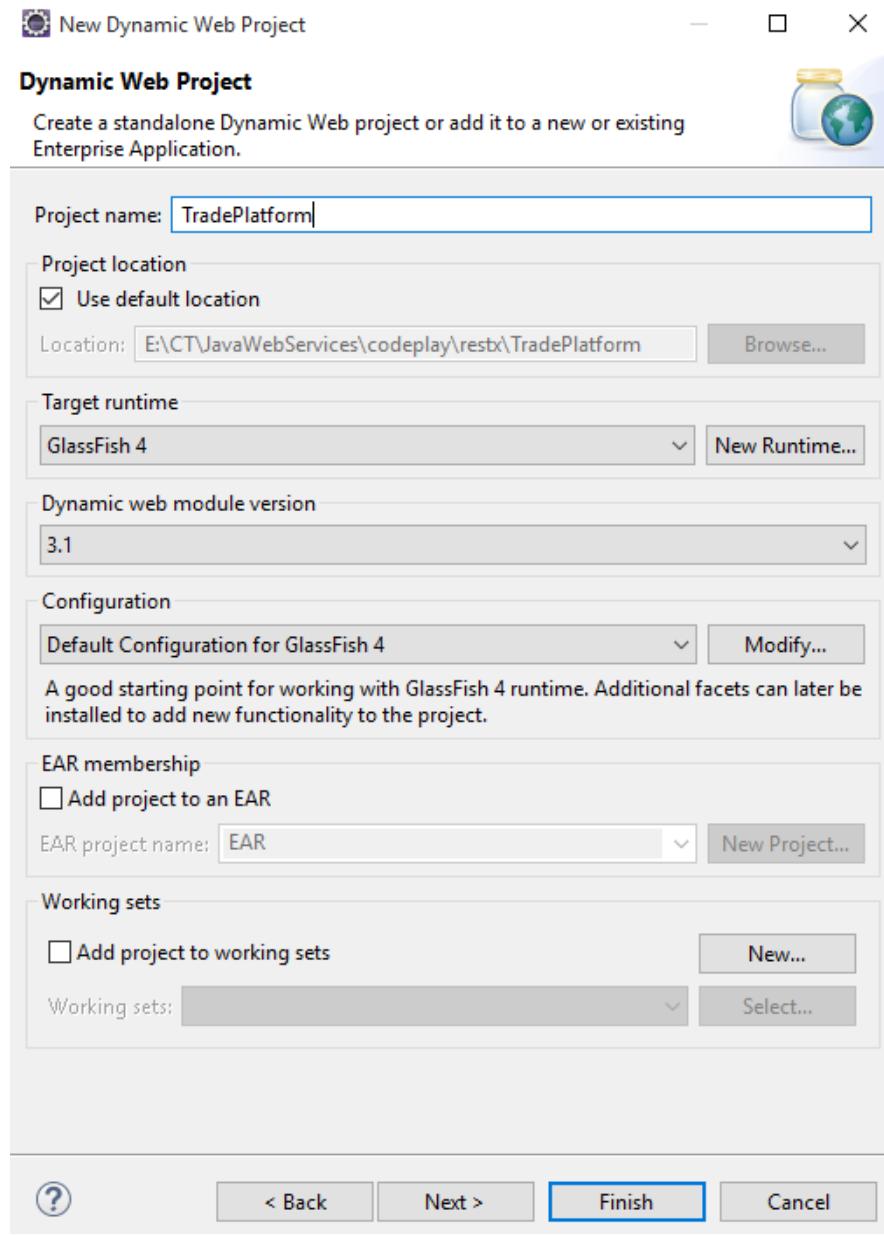


SYED AWASE

VII(C): CONTRACT FIRST (WSDL FIRST) APPROACH TO CREATE A WS ENDPOINT: EXAMPLE 2

Step 1

- create a Dynamic Web Project with target runtime GlassFish 4



Step 2

- create a skeleton of StockServiceImpl() Service

```
package com.sycliq.business;

import java.util.List;

public class StockServiceImpl {
    public StockServiceImpl() {
    }

    public List<String> getStockCategories(){
        return null;
    }

    public List<String> getStocks(String stockCategory){
        return null;
    }

    public boolean addStock(String stockCategory, String stock) {
        return true;
    }
}
```



Step 3

- create basic StockCatalog with appropriate methods to fetch data

```
package com.sycliq.trade;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

import com.sycliq.business.StockServiceImpl;

@WebService
public class StockCatalog {

    StockServiceImpl stockService = new StockServiceImpl();

    @WebMethod
    public List<String> getStockCategories(){
        return stockService.getStockCategories();
    }

    @WebMethod
    public List<String> getStocks(String category){

        return stockService.getStocks(category);
    }

    @WebMethod
    public boolean addStock(String category, String stock) {

        return true;
    }
}
```



Step 4

- basic wsdl generated post deployment of application on to glassfish web server
- Customize the WSDL to suit your requirements.

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown. --><!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown. --><definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://trade.sycliq.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.sycliq.com/" name="StockCatalogService"><types><message name="addStock"></message><message name="addStockResponse"></message><message name="getStocks"></message><message name="getStocksResponse"></message><message name="getStockCategories"></message><message name="getStockCategoriesResponse"></message><portType name="StockCatalog"></portType><binding name="StockCatalogPortBinding" type="tns:StockCatalog"><service name="StockCatalogService"><port name="StockCatalogPort" binding="tns:StockCatalogPortBinding"><soap:address location="http://desktop\_1n4b4y1:9090/TradePlatform/StockCatalogService" /></port></service></definitions>
```



Step 5

- **@WebMethod(exclude=true)**
- This is used to exclude a method from the WSDL generated.

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://trade.syqliq.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.syqliq.com/" name="StockCatalogService">
  <types>
    ><xsd:schema>...</xsd:schema>
  </types>
  ><message name="getStockCategories">...</message>
  ><message name="getStockCategoriesResponse">...</message>
  ><portType name="StockCatalog">...</portType>
  ><binding name="StockCatalogPortBinding" type="tns:StockCatalog">...</binding>
  ><service name="StockCatalogService">
    ><port name="StockCatalogPort" binding="tns:StockCatalogPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockCatalogService"/>
    </port>
  </service>
</definitions>
```

} Observe that only getStockCategories() method is rendered



Step 6

- default it takes the name of the class as the **portType's name**.
- **@WebService(name="SycliQStockCatalog")** overrides the default name

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns: wsp="http://www.w3.org/ns/ws-policy" xmlns: wspi_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns: wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns: soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns: tns="http://trade.sycliq.com/" xmlns: xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.sycliq.com/" name="StockCatalogService">
  <types>
    <xsd:schema>..</xsd:schema>
  </types>
  <message name="getStockCategories">...</message>
  <message name="getStockCategoriesResponse">...</message>
  <portType name="StockCatalog">
    <operation name="getStockCategories">
      <input wsam:Action="http://trade.sycliq.com/StockCatalog/getStockCategoriesRequest" message="tns:getStockCategories"/>
      <output wsam:Action="http://trade.sycliq.com/StockCatalog/getStockCategoriesResponse" message="tns:getStockCategoriesResponse"/>
    </operation>
    <binding name="StockCatalogPortBinding" type="tns:StockCatalog">...</binding>
  </portType>
  <service name="StockCatalogService">
    <port name="StockCatalogPort" binding="tns:StockCatalogPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockCatalogService"/>
    </port>
  </service>
</definitions>
```

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns: wsp="http://www.w3.org/ns/ws-policy" xmlns: wspi_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns: wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns: soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns: tns="http://trade.sycliq.com/" xmlns: xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.sycliq.com/" name="StockCatalogService">
  <types>...</types>
  <message name="getStockCategories">
    <part name="parameters" element="tns:getStockCategories"/>
  </message>
  <message name="getStockCategoriesResponse">
    <part name="parameters" element="tns:getStockCategoriesResponse"/>
  </message>
  <portType name="SycliQStockCatalog">
    <operation name="getStockCategories">
      <input wsam:Action="http://trade.sycliq.com/SycliQStockCatalog/getStockCategoriesRequest" message="tns:getStockCategories"/>
      <output wsam:Action="http://trade.sycliq.com/SycliQStockCatalog/getStockCategoriesResponse" message="tns:getStockCategoriesResponse"/>
    </operation>
  </portType>
  <binding name="SycliQStockCatalogPortBinding" type="tns:SycliQStockCatalog">...</binding>
  <service name="StockCatalogService">
    <port name="SycliQStockCatalogPort" binding="tns:SycliQStockCatalogPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockCatalogService"/>
    </port>
  </service>
</definitions>
```



Step 7

- use **ctrl+space** to list the properties that can be added to `@WebService()`
- Similarly the properties that are available for `@WebMethod()`

The screenshot shows an IDE interface with Java code. A tooltip is displayed over the `endpointInterface` field of the `@WebService` annotation. The tooltip contains the following text:
The complete name of the service endpoint interface defining the service's abstract Web Service contract. This annotation allows the developer to separate the interface contract from the implementation. If this annotation is present, the service endpoint interface is used to determine the abstract WSDL contract (portType and bindings). The service endpoint interface MAY include JSR-181 annotations to customize the mapping from Java to WSDL.
The service implementation bean MAY implement the service
Press 'Tab' from proposal table or click for focus

```
import com.sycliq.business.StockServiceImpl;

@WebService(name="SycliqStockCatalog", portName="SycliqStockCatalogPort")
public class StockCatalog {

    StockServiceImpl stockService = new StockServiceImpl();

    @WebMethod(operationName="getStocks")
    public List<String> getStocks() {
        return stockService.getStocks();
    }

    @WebMethod(operationName="addStock")
    public List<String> addStock(String stock) {
        return stockService.addStock(stock);
    }

    @WebMethod(operationName="removeStock")
    public boolean removeStock(String stock) {
        return true;
    }
}
```

```
@WebService(name="SycliqStockCatalog", portName="SycliqStockCatalogPort")
public class StockCatalog {
```

The screenshot shows an IDE interface with Java code. A tooltip is displayed over the `action` field of the `@WebMethod` annotation. The tooltip contains the following text:
The action name for the Web Method. This is the name of the operation as it appears in the WSDL.
The action name for the Web Method. This is the name of the operation as it appears in the WSDL.
The action name for the Web Method. This is the name of the operation as it appears in the WSDL.
The action name for the Web Method. This is the name of the operation as it appears in the WSDL.

```
    @WebMethod(operationName="getStocks")
    public List<String> getStocks() {
        return stockService.getStocks();
    }

    @WebMethod(operationName="addStock")
    public List<String> addStock(String stock) {
        return stockService.addStock(stock);
    }

    @WebMethod(operationName="removeStock")
    public boolean removeStock(String stock) {
        return true;
    }
}
```



Step 8:

- Add properties for
**@WebMethod(action="fetch_stockcategories",
operationName="fetchStockCategories")**

```
@WebService(name="SycliQStockCatalog", portName="SycliQStockCatalogPort")
public class StockCatalog {

    StockServiceImpl stockService = new StockServiceImpl();

    @WebMethod(action="fetch_stockcategories", operationName="fetchStockCategories" )
    public List<String> getStockCategories(){
        return stockService.getStockCategories();
    }
    @WebMethod(exclude=true)
    public List<String> getStocks(String category){

        return stockService.getStocks(category);
    }
    @WebMethod(exclude=true)
    public boolean addStock(String category, String stock) {

        return true;
    }
}

<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns: wsp="http://www.w3.org/ns/ws-policy" xmlns: wspl_2="http://www.w3.org/ns/ws-policy#"
    xmlns: wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns: soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns: tns="http://trade.sycliq.com/" xmlns: xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.sycliq.com/" name="StockCatalogService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://trade.sycliq.com/" schemaLocation="http://desktop_1n4b4y1:9090/TradePlatform/StockCatalogService?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="fetchStockCategories">
        <part name="parameters" element="tns:fetchStockCategories"/>
    </message>
    <message name="fetchStockCategoriesResponse">
        <part name="parameters" element="tns:fetchStockCategoriesResponse"/>
    </message>
    <portType name="SycliQStockCatalog">
        <operation name="fetchStockCategories">
            <input wsam:Action="fetch_stockcategories" message="tns:fetchStockCategories"/>
            <output wsam:Action="http://trade.sycliq.com/SycliQStockCatalog/fetchStockCategoriesResponse" message="tns:fetchStockCategoriesResponse"/>
        </operation>
    </portType>
    <binding name="SycliQStockCatalogPortBinding" type="tns:SycliQStockCatalog">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="fetchStockCategories">
            <soap:operation soapAction="fetch_stockcategories"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="StockCatalogService">
        <port name="SycliQStockCatalogPort" binding="tns:SycliQStockCatalogPortBinding">
            <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockCatalogService"/>
        </port>
    </service>
</definitions>
```



Step 9

- create StockInfo class with WebService and WebMethod annotation

```
@WebService(name="SycliQStockCatalog", portName="SycliQStockCatalogPort")
public class StockCatalog {

    StockServiceImpl stockService = new StockServiceImpl();

    @WebMethod(action="fetch_stockcategories", operationName="fetchStockCategories")
    public List<String> getStockCategories(){
        return stockService.getStockCategories();
    }
    @WebMethod(exclude=true)
    public List<String> getStocks(String category){

        return stockService.getStocks(category);
    }
    @WebMethod(exclude=true)
    public boolean addStock(String category, String stock) {

        return true;
    }
}
```



Step 10

- wsdl generated on publishing to glassfish server

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://trade.syqliq.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.syqliq.com/" name="StockInfoService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://trade.syqliq.com/" schemaLocation="http://desktop_1n4b4y1:9090/TradePlatform/StockInfoService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getStockInfo">
    <part name="parameters" element="tns:getStockInfo"/>
  </message>
  <message name="getStockInfoResponse">
    <part name="parameters" element="tns:getStockInfoResponse"/>
  </message>
  <portType name="StockInfo">
    <operation name="getStockInfo">
      <input wsam:Action="http://trade.syqliq.com/StockInfo/getStockInfoRequest" message="tns:getStockInfo"/>
      <output wsam:Action="http://trade.syqliq.com/StockInfo/getStockInfoResponse" message="tns:getStockInfoResponse"/>
    </operation>
  </portType>
  <binding name="StockInfoPortBinding" type="tns:StockInfo">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getStockInfo">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="StockInfoService">
    <port name="StockInfoPort" binding="tns:StockInfoPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockInfoService"/>
    </port>
  </service>
</definitions>
```



Step 11

- xsd schema reference inside the types element in the WSDL

```
<xs:schema xmlns:tns="http://trade.sycliq.com/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0" targetNamespace="http://trade.sycliq.com/">  
  <xs:element name="getStockInfo" type="tns:getStockInfo"/>  
  <xs:element name="getStockInfoResponse" type="tns:getStockInfoResponse"/>  
  ▼<xs:complexType name="getStockInfo">  
    ▼<xs:sequence>  
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>  
    </xs:sequence>  
  </xs:complexType>  
  ▼<xs:complexType name="getStockInfoResponse">  
    ▼<xs:sequence>  
      <xs:element name="return" type="xs:string" minOccurs="0"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:schema>
```



Step 12

- @SOAPBinding by default style is document type literal, which gives the types and a corresponding XSD as shown in step 11.
- @SOAPBinding(style=Style.RPC)
- ***In this style, the schema is not available to be validated!***

```
package com.sycliq.trade;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding(style=Style.RPC) ←
public class StockInfo {

    @WebMethod
    public String getStockInfo(String property) {
        String response = "Invalid Property";
        if("stockName".equals(property)) {
            response="Sycliq";
        }
        else if("month".equals(property)) {
            response="January";
        }
        return response;
    }

}
```



Step 13

- On publishing the service to glassfish, the generated wsdl file looks bit different, unlike in Step 10.

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://trade.sycliq.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.sycliq.com/" name="StockInfoService">
  <types/>
    <message name="getStockInfo">
      <part name="arg0" type="xsd:string"/>
    </message>
    <message name="getStockInfoResponse">
      <part name="return" type="xsd:string"/>
    </message>
  <portType name="StockInfo">
    <operation name="getStockInfo">
      <input wsam:Action="http://trade.sycliq.com/StockInfo/getStockInfoRequest" message="tns:getStockInfo"/>
      <output wsam:Action="http://trade.sycliq.com/StockInfo/getStockInfoResponse" message="tns:getStockInfoResponse"/>
    </operation>
  </portType>
  <binding name="StockInfoPortBinding" type="tns:StockInfo">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="getStockInfo">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" namespace="http://trade.sycliq.com/" />
      </input>
      <output>
        <soap:body use="literal" namespace="http://trade.sycliq.com/" />
      </output>
    </operation>
  </binding>
  <service name="StockInfoService">
    <port name="StockInfoPort" binding="tns:StockInfoPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockInfoService"/>
    </port>
  </service>
</definitions>
```



Step 14

- @WebParam(partName=“inputData”)
- @WebResult(partName=“outputData”)
- For input request and output response

```
package com.sycliq.trade;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding(style=Style.RPC)
public class StockInfo {

    @WebMethod
    @WebResult(partName="outputData")
    public String getStockInfo(@WebParam(partName="inputData") String property) {
        String response ="Invalid Property";
        if("stockName".equals(property)) {
            response="SycliQ";
        }
        else if("month".equals(property)) {
            response="January";
        }
        return response;
    }
}
```



Step 15

- post publishing and rendering of the wsdl file in glassfish server

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns: wsp="http://www.w3.org/ns/ws-policy" xmlns: wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns: wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns: soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns: tns="http://trade.syqliq.com/" xmlns: xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://trade.syqliq.com/" name="StockInfoService">
  <types>
    <message name="getStockInfo">
      <part name="inputData" type="xsd:string"/>
    </message>
    <message name="getStockInfoResponse">
      <part name="outputData" type="xsd:string"/>
    </message>
  </types>
  <portType name="StockInfo">
    <operation name="getStockInfo">
      <input wsam:Action="http://trade.syqliq.com/StockInfo/getStockInfoRequest" message="tns:getStockInfo"/>
      <output wsam:Action="http://trade.syqliq.com/StockInfo/getStockInfoResponse" message="tns:getStockInfoResponse"/>
    </operation>
  </portType>
  <binding name="StockInfoPortBinding" type="tns:StockInfo">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="getStockInfo">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" namespace="http://trade.syqliq.com/" />
      </input>
      <output>
        <soap:body use="literal" namespace="http://trade.syqliq.com/" />
      </output>
    </operation>
  </binding>
  <service name="StockInfoService">
    <port name="StockInfoPort" binding="tns:StockInfoPortBinding">
      <soap:address location="http://desktop_1n4b4y1:9090/TradePlatform/StockInfoService"/>
    </port>
  </service>
</definitions>
```





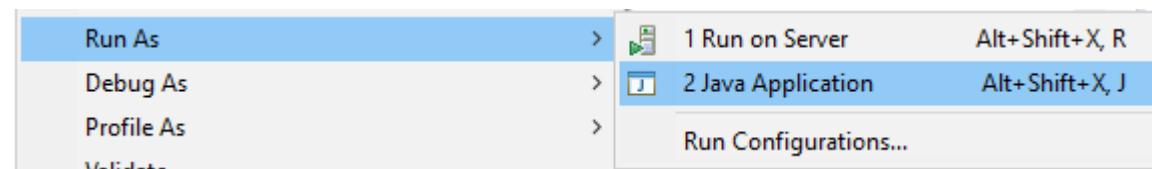
SYED AWASE

USING ENDPOINT TO RENDER WEB SERVICE (METRO) WITHOUT GLASSFISH SERVER

Step 1

- create a class StockEndpointPublisher to render an endpoint directly without using glassfish server.
- Run as a java application

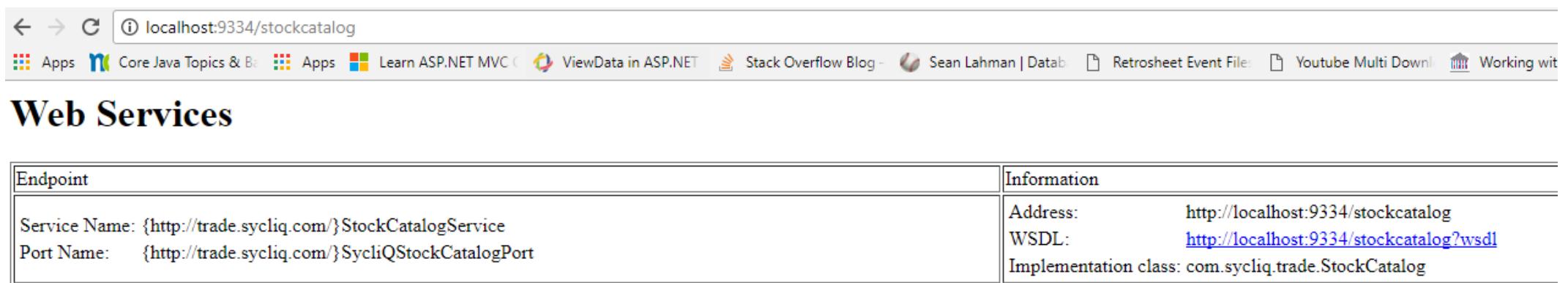
```
package com.syqliq.trade;  
  
import javax.xml.ws.Endpoint;  
  
public class StockEndpointPublisher {  
    public static void main(String[] args) {  
        Endpoint.publish("http://localhost:9334/stockcatalog", new StockCatalog());  
    }  
}
```



Step 2

- open the rendered endpoint service at

<http://localhost:9334/stockcatalog>



The screenshot shows a web browser window with the URL `localhost:9334/stockcatalog` in the address bar. The page content is titled "Web Services" and displays the following table:

Endpoint	Information
Service Name: { http://trade.syqliq.com/ } StockCatalogService Port Name: { http://trade.syqliq.com/ } SycliQStockCatalogPort	Address: http://localhost:9334/stockcatalog WSDL: http://localhost:9334/stockcatalog?wsdl Implementation class: com.syqliq.trade.StockCatalog





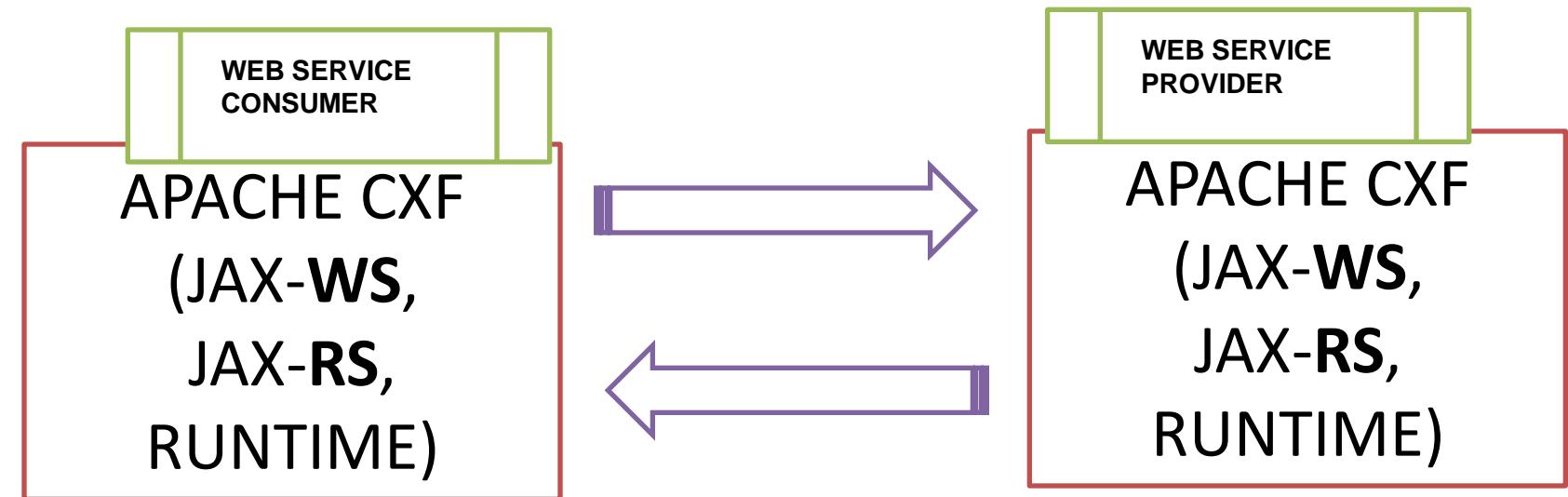
SYED AWASE

VIII: JAX-WS WITH APACHE CXF

JAX-WS

- Apache CXF
- Apache AXIS
- Metro/Glassfish
- WebLogic
- WebSphere

WEB SERVICES ENGINE



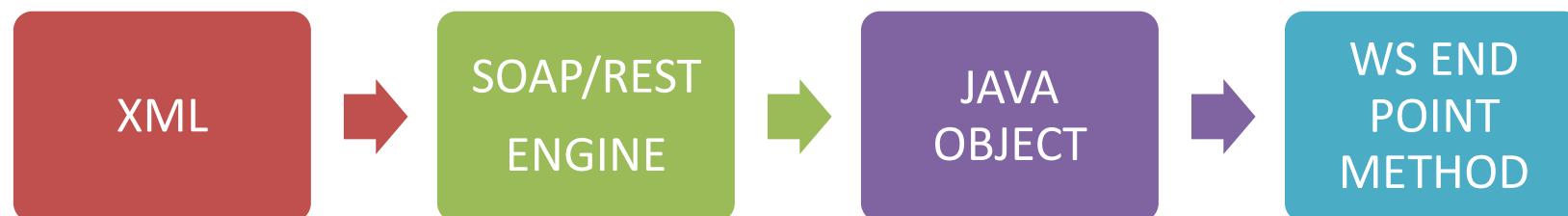


APACHE CXF

APACHE CXF?

v3.0.3

- SOAP/REST ENGINE
 - Serialize and De-Serialize
 - Publish and Dispatch
- Tools
 - Wsdl2java
 - Java2wsdl
- Spring Configuration
- Implements Web Service Standards
 - WS-Transaction Management
 - WS-Security
 - WS-Policy
- Easy to customize using Interceptors and Handlers.



<http://cxf.apache.org/>

© TPRI Syed Awase 2013-14 - Java Web Services Ground Up!



Why Apache CXF?

- Easy customization of loggers
- Inbound and outbound interceptor
- Powerful mechanism for application security
- Easy exception handling using custom fault
- Ease and simple integration with spring framework.
- Provides tools to work with schema/WSDL
- Flexibility in deployment in web container.
- Clean separation of java code and JAX-WS



INTERCEPTOR DESIGN PATTERN

- A software design pattern used when software systems or frameworks **want to offer a way to change or augment, their usual processing cycle.**
- **The change is transparent and used automatically**
- **Most widely used in web-servers, object and message-oriented middleware.**
- Interceptors are commonly used when a business method on a target class is invoked on lifecycle events **to create/destroy or timeout**, such as found when **implementing profiling, auditing, logging and so forth.**
- **Monitor what the application is doing inside**
- **Provide possibility to change some of the application's behavior.**

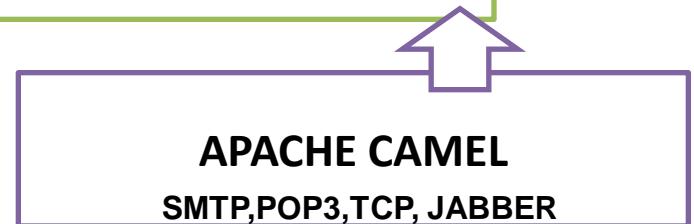


INTERCEPTOR DESIGN PATTERN

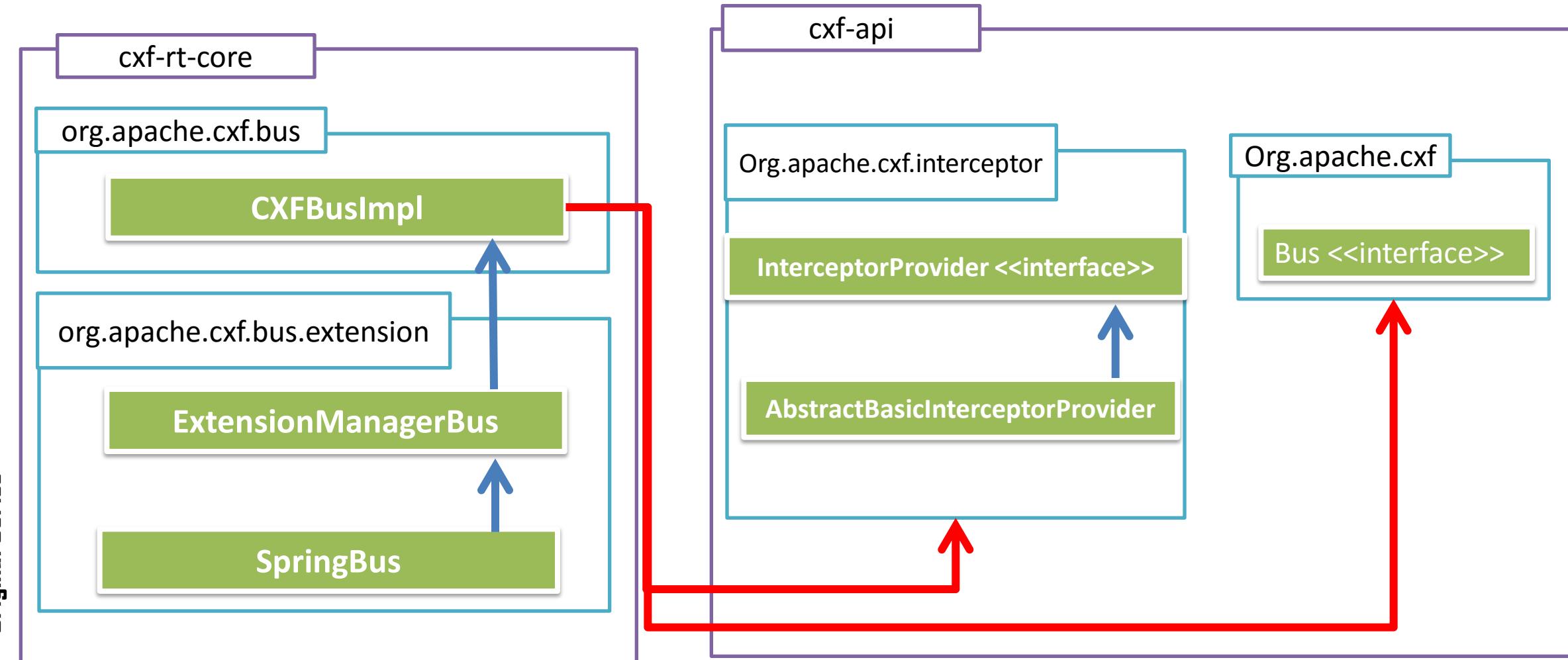
- It helps to extend application functionality with new features
 - Implementer of the extension need not know rest of the application.
 - Implementer of the extension need not change existing code
 - New extensions can or cannot affect current system.
- Mostly used to intercepts the requests by a client to a specific resource.



APACHE CXF SERVICES FRAMEWORK



APACHE CXF BUS CLASS DIAGRAM

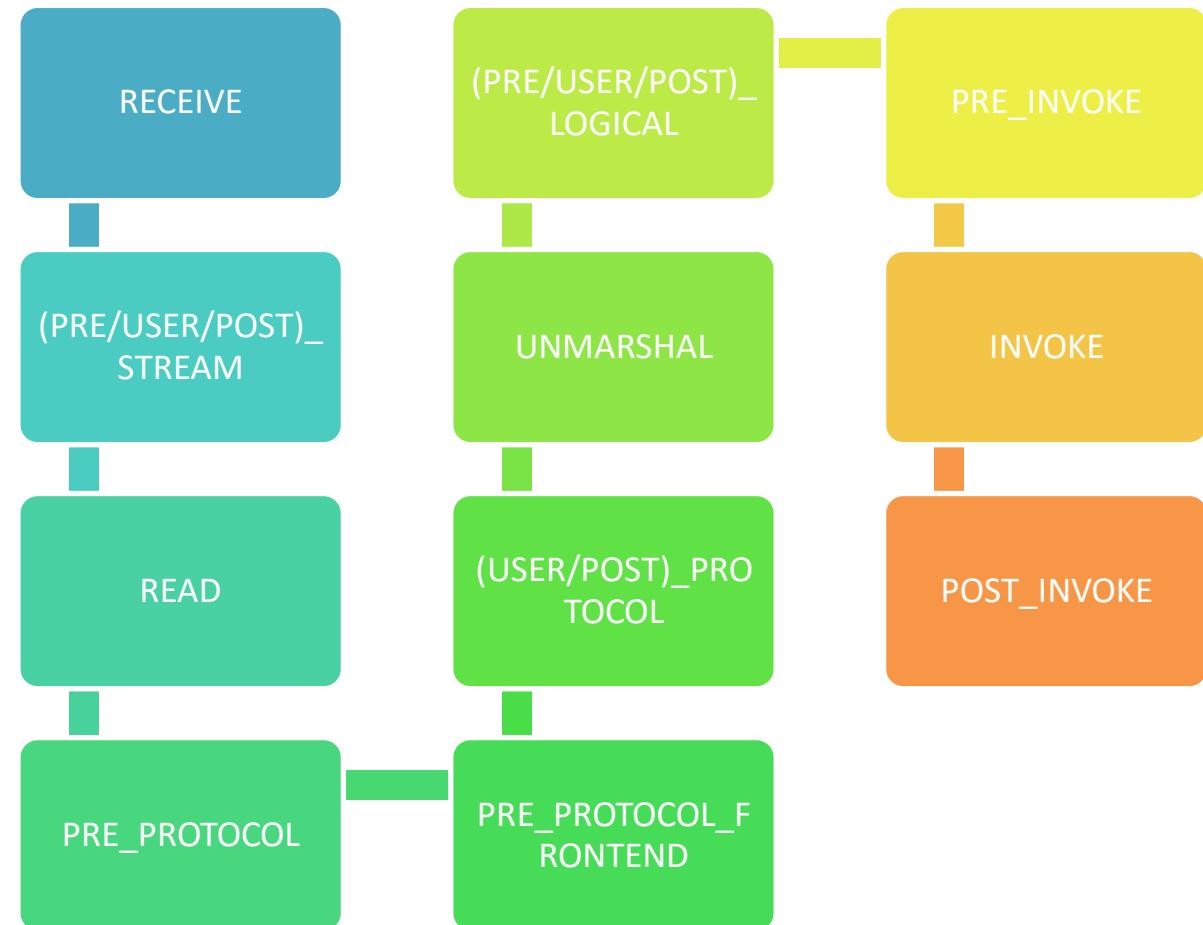


INTERCEPTORS AND MESSAGES

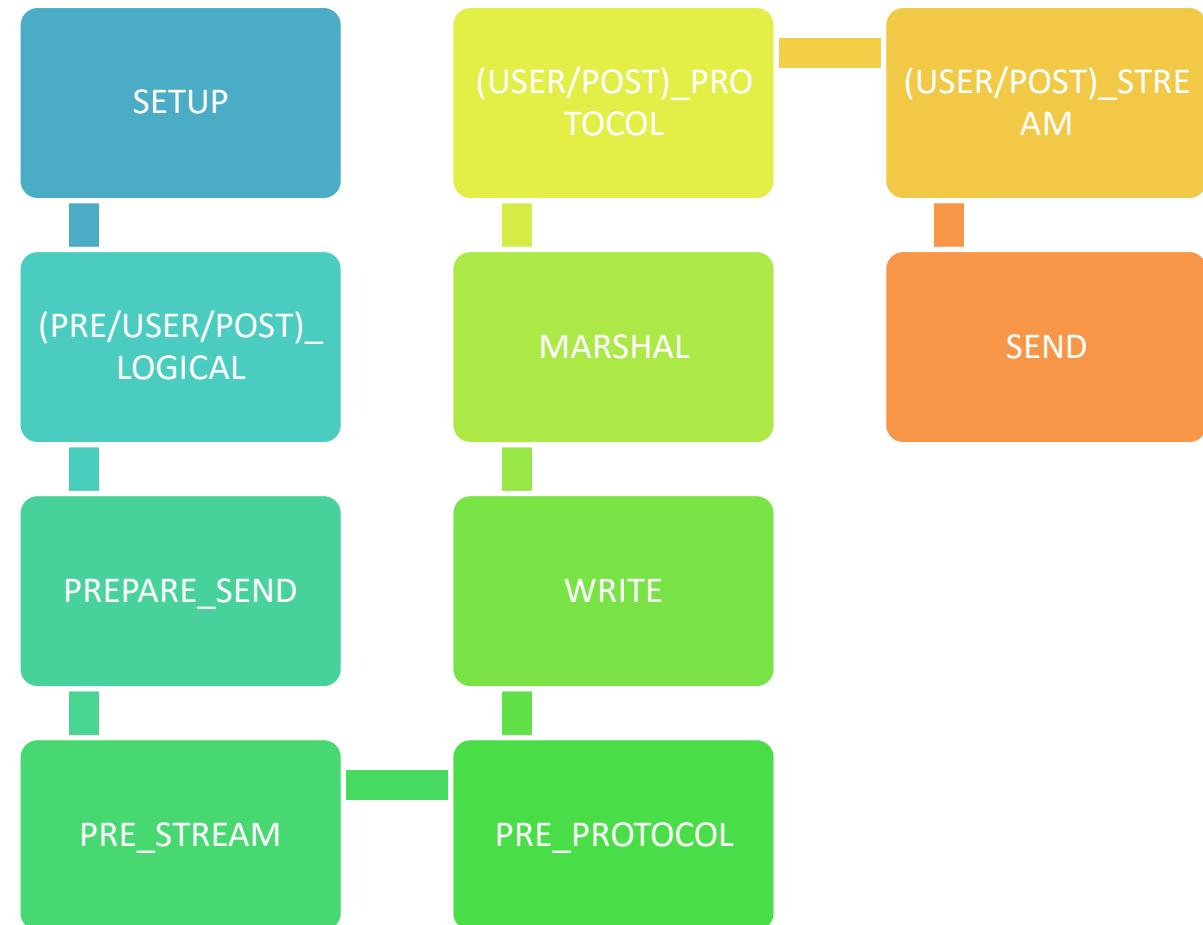
- Interceptors
 - Follows interceptor design pattern
 - Provides message and fault handlers
 - Can be chained together
- CXF uses interceptor phases to logically identify where an interceptor chain belongs to interceptor process
 - Incoming chains
 - Outgoing chains
 - Post-processing chains
- Messages
 - Container for data to be passed through interceptor chains.



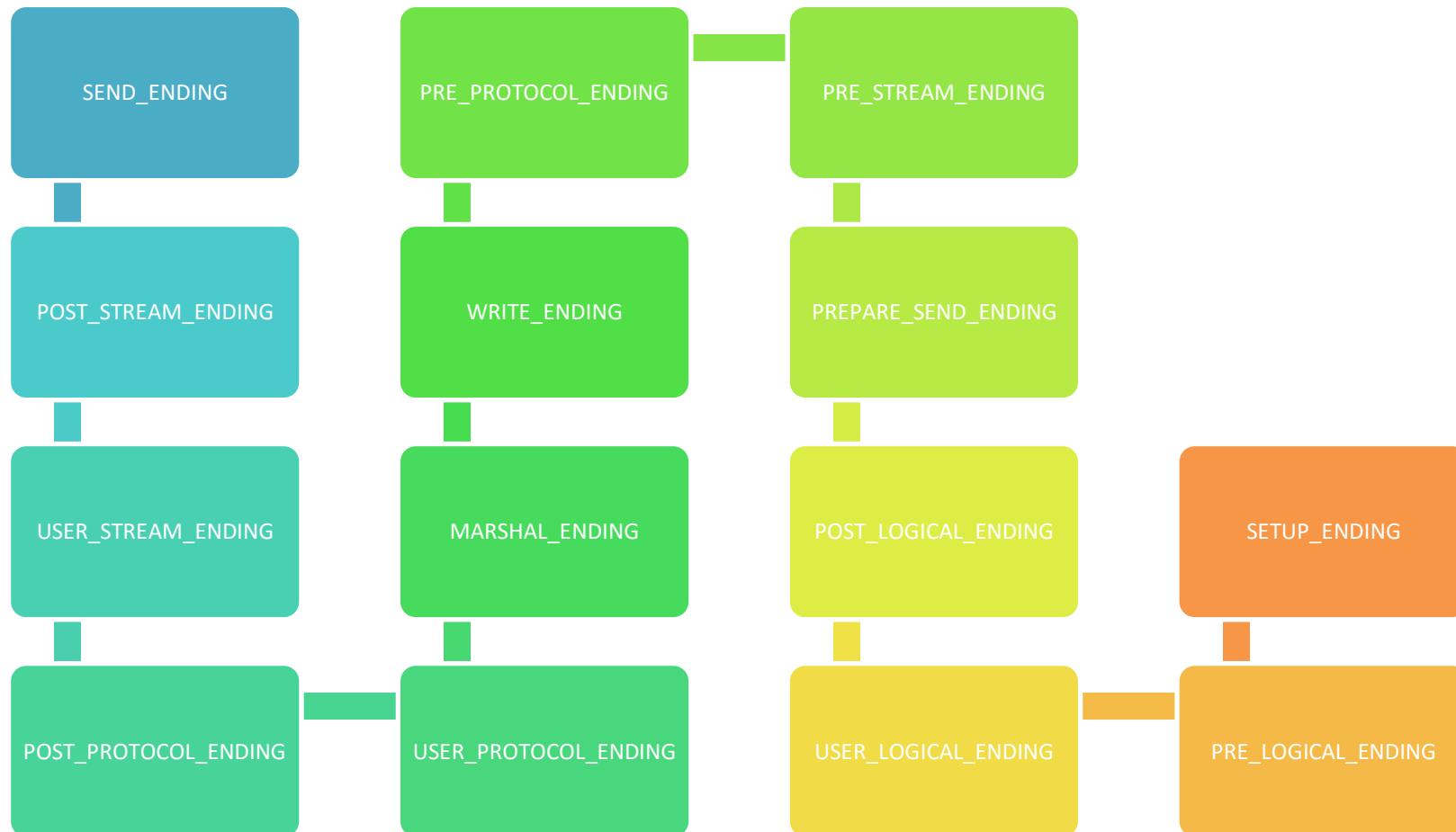
PHASES FOR INCOMING INTERCEPTOR CHAINS



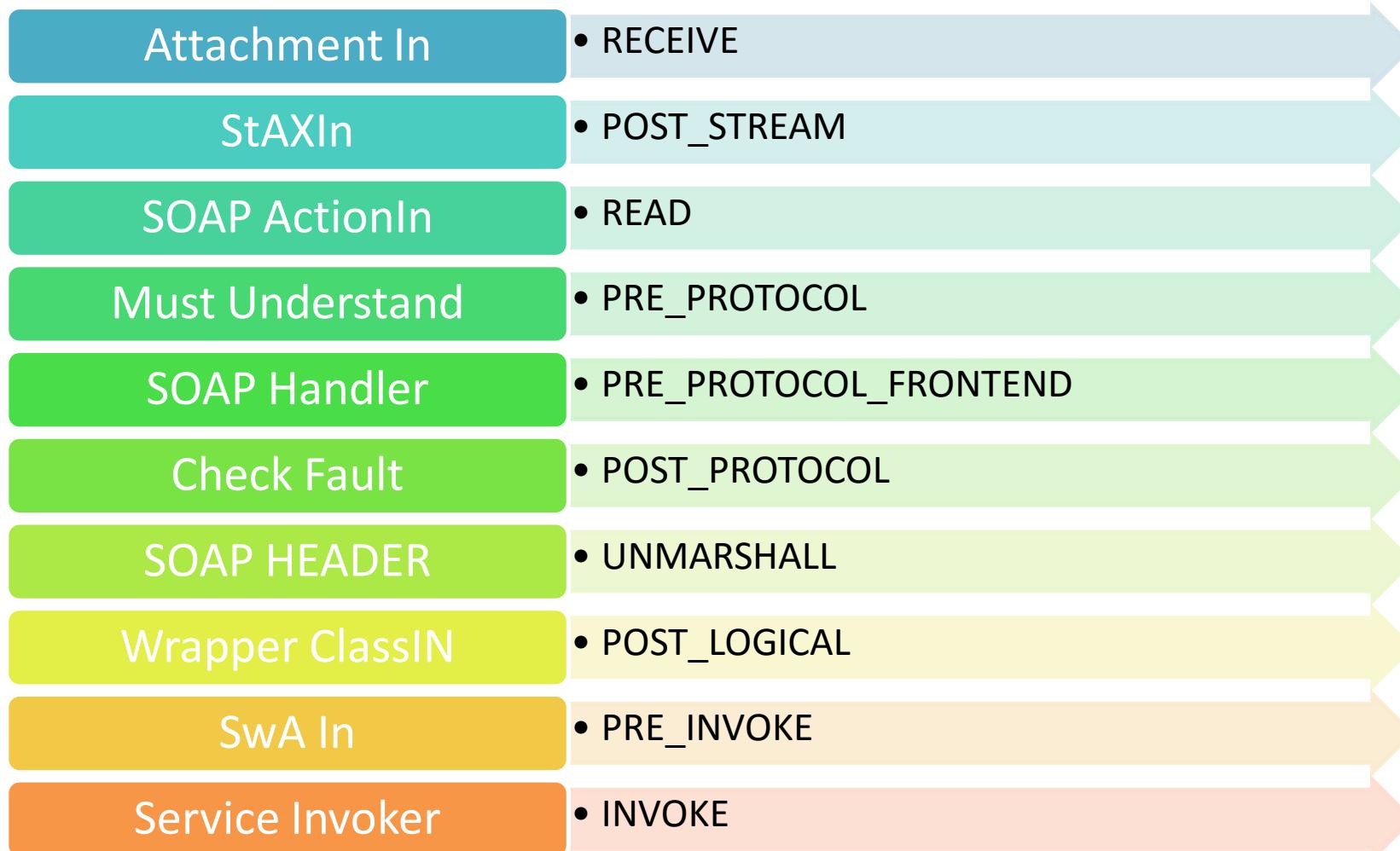
PHASES FOR OUTGOING INTERCEPTOR CHAINS



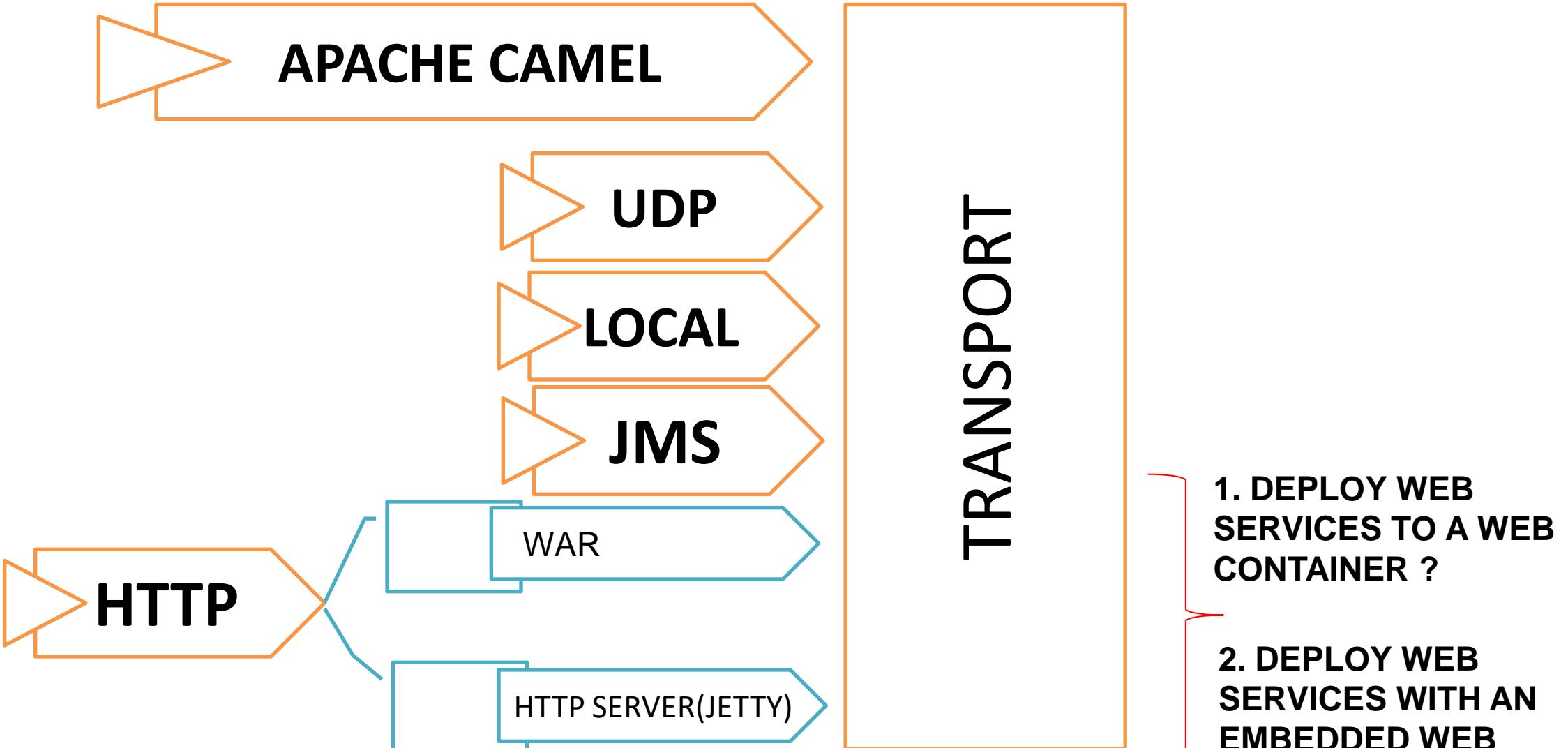
PHASES FOR POST-PROCESSING INTERCEPTOR CHAINS



JAX-WS INTERCEPTOR CHAIN



APACHE CXF TRANSPORT OPTIONS



APACHE CXF WEB CONTAINER SUPPORT

NOT FULL JAVA EE CONTAINERS

APACHE TOMCAT

JETTY

JBOSS
APPLICATION
SERVER

WEBLOGIC

WEBSPHERE
APPLICATION
SERVER

GLASSFISH
APPLICATION
SERVER



APACHE CXF FRONT END OPTIONS

-  **JAVASCRIPT**
-  **JAX-RS**
-  **JAX-WS**

FONT-END OPTIONS



Apache CXF Maven Archetypes

- Apache CXF currently offers two Archetype for each front end
 - JAX-WS Archetype
 - It includes java-first web service
 - Command to generate a WS archetype
 - mvn archetype: generate – Dfilter=org.apache.cxf.archetype
 - JAX-RS Archetype

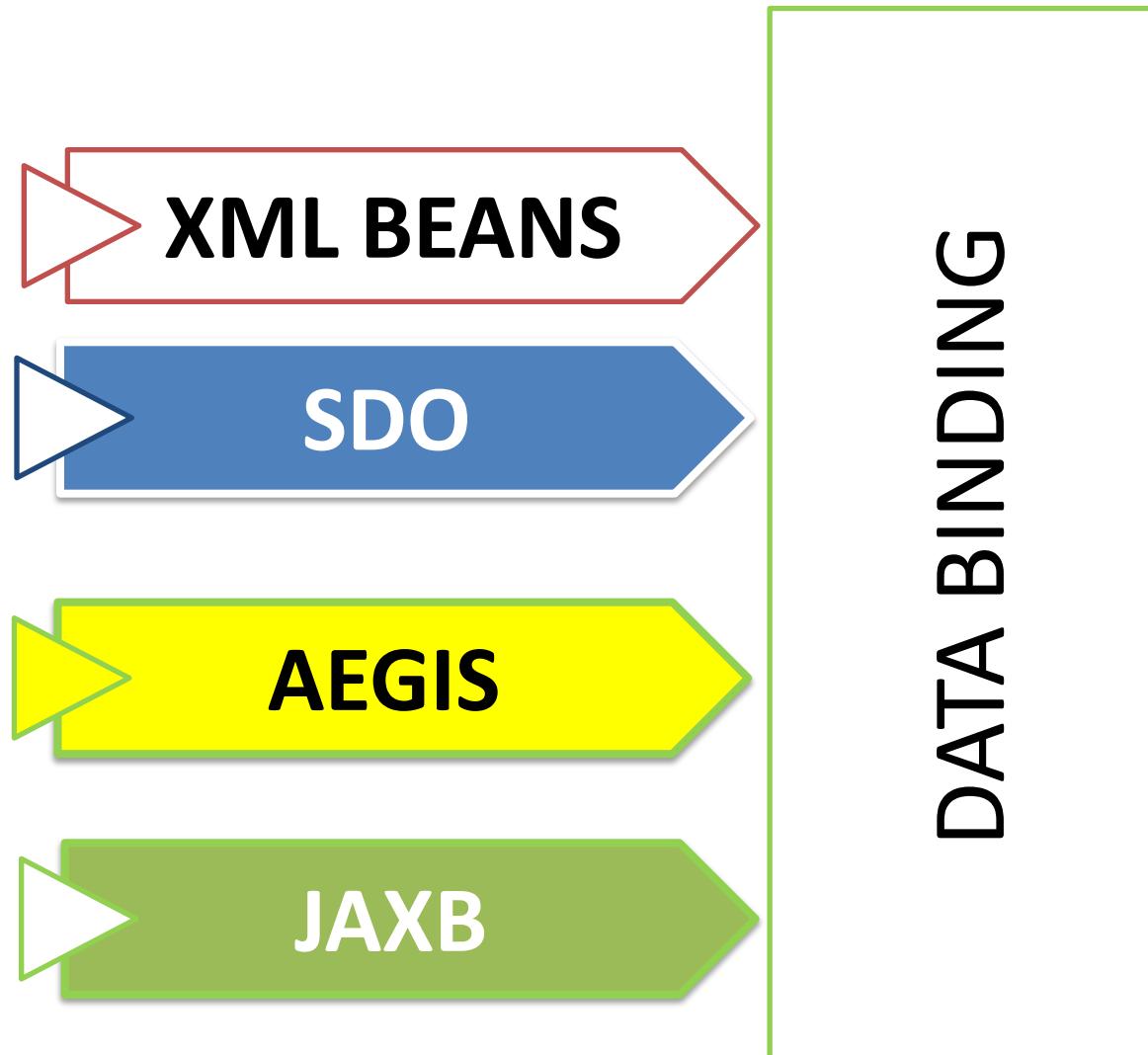


JAX-WS

- Specification for Java XML-based web services
- Typically implemented using a WSDL with SOAP over Http
- CXF provides full support for JAX-WS



APACHE CXF DATA BINDING OPTIONS



JAXB

- Java architecture for XML binding.
- Binding based on XML schema definition
- Unmarshal and marshal
- Uses
com.apache.cxf.jaxb.JAXBDataBinding



APACHE CXF XJC MAVEN PLUGIN

- XJC is a binding compiler executed through a command prompt
- Generate java code based on an XSD
- Apache CXF supports XJC through a Maven plug-in



APACHE CXF PROTOCOL BINDING OPTIONS

-  **PURE XML**
-  **MTOM***
-  **SOAP**

PROTOCOL BINDING

*MTOM: SOAP MESSAGE TRANSMISSION AND OPTIMISATION MECHANISM (SENDING BINARY DATA AS SOAP MSG)



SOAP PROTOCOL BINDING

- Language that defines service message format
- Data is passed in an envelope that contains a header and body
- Configured as part of the WSDL **BINDING** section.



WS-* SPECIFICATION OPTIONS

ADDRESSING

- Standard way of adding address information to a SOAP header

DISCOVERY (UDP)

- Support for multicast protocol that auto discovers services on local network

METADATA EXCHANGE

- How metadata is represented for a web service end point
- cxf.rt-ws-mex

POLICY

- Framework and model for web service policies
- Cxf-rt-ws-policy

RELIABLE MESSAGING

- Protocol for reliable message delivery between distributed systems

SECURE CONVERSATION

- Provides security features beyond the transport level protocol
- Cxf-rt-ws-security

SECURITY

- Provides easier, standards-based way to configure security

SECURITY POLICY

- Configure security policy

TRUST

- Supports the issuing, renewing and validation of security tokens

WS-*SPECIFICATION
OPTIONS



JAX-WS VS JAX-RS :CHOICE??

JAX-WS

- Used for distributed component integration within enterprise or different vendors
- Complex operations (task services, entity services and utility services)
- Standards-based (SOAP,XML,WSDL)
- Multiple transports (support for web services over JMS)

JAX-RS

- Makes the most sense when integrating with Mobile and web views.
- Reduced overhead in terms of message processing and communications
- Simple transactions (CRUD Operations mapping to Http Action Verbs)
- More flexibility with limited constraints
- Multiple data formats (JSON,XML,HTML,PlainText, binary)
- HTTP transport



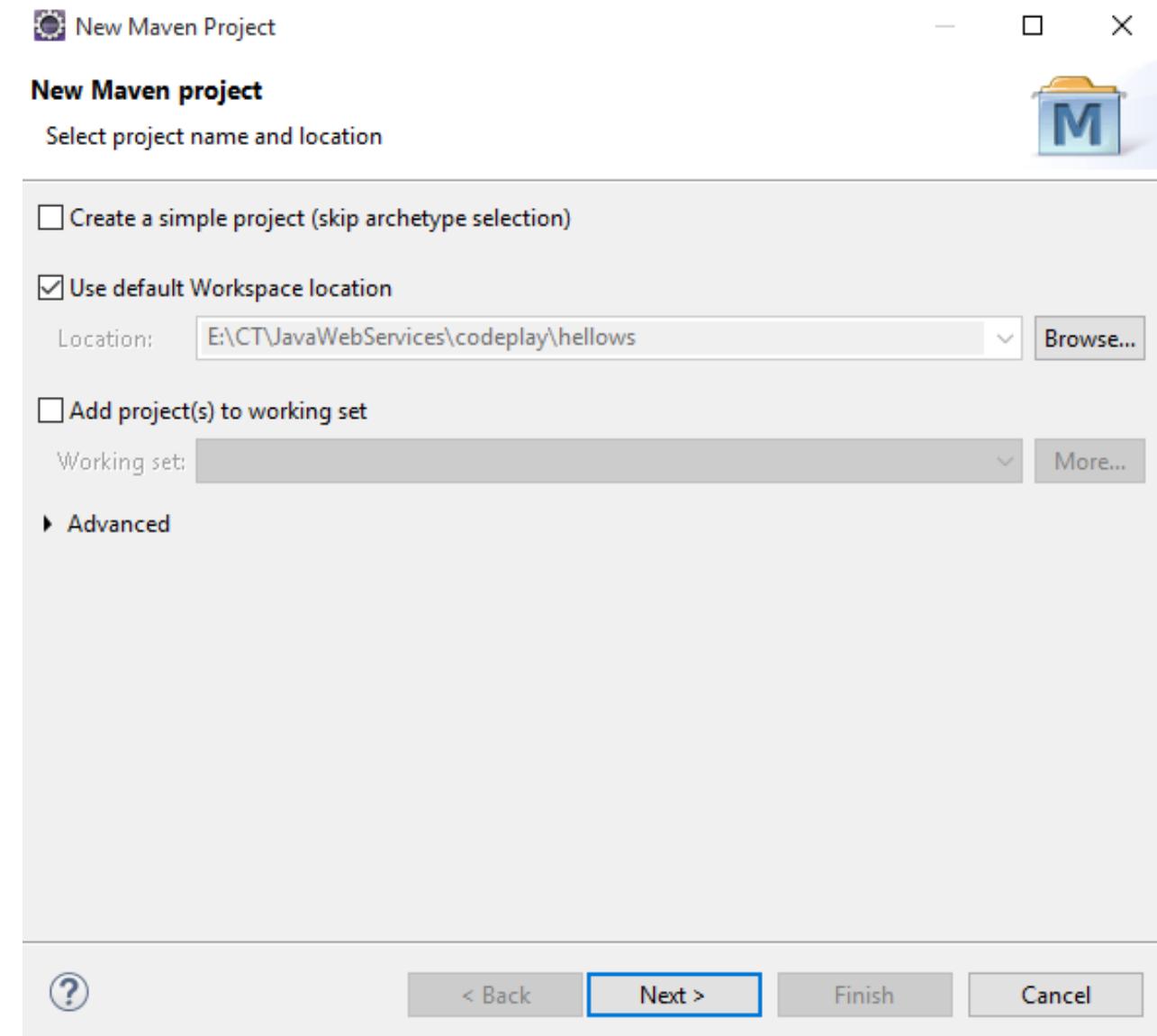


APACHE CXF

VIII(A):PLAY BOOK: SIMPLE WEB SERVICE (JAX-WS)

STEP 1

- **CREATE A NEW MAVEN PROJECT USING DEFAULT WORKSPACE LOCATION.**

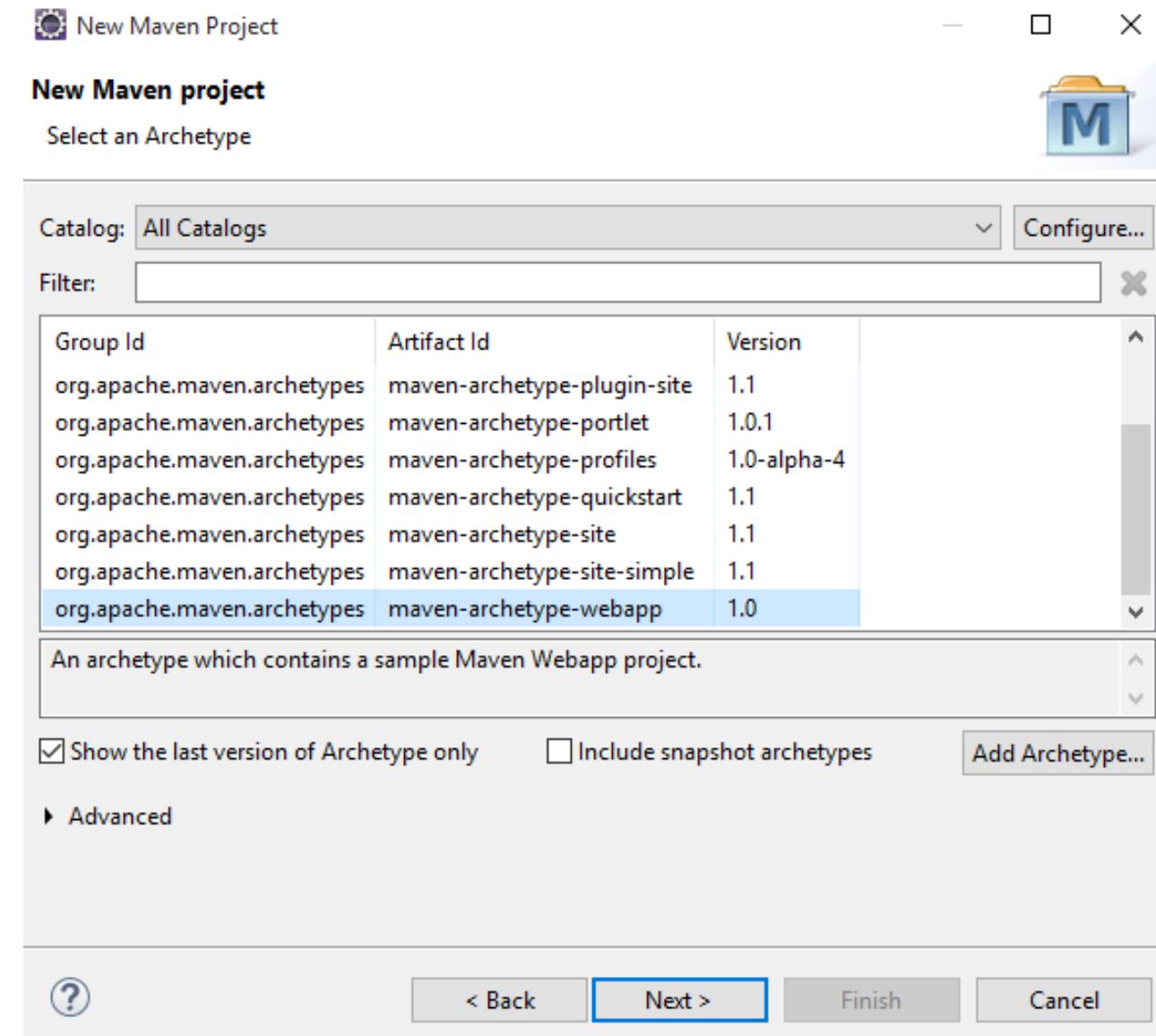


STEP 2

- SELECT
ORG.APACHE.MAVEN.
ARCHETYPES =>
**MAVEN.ARCHETYPE-
WEBAPP**

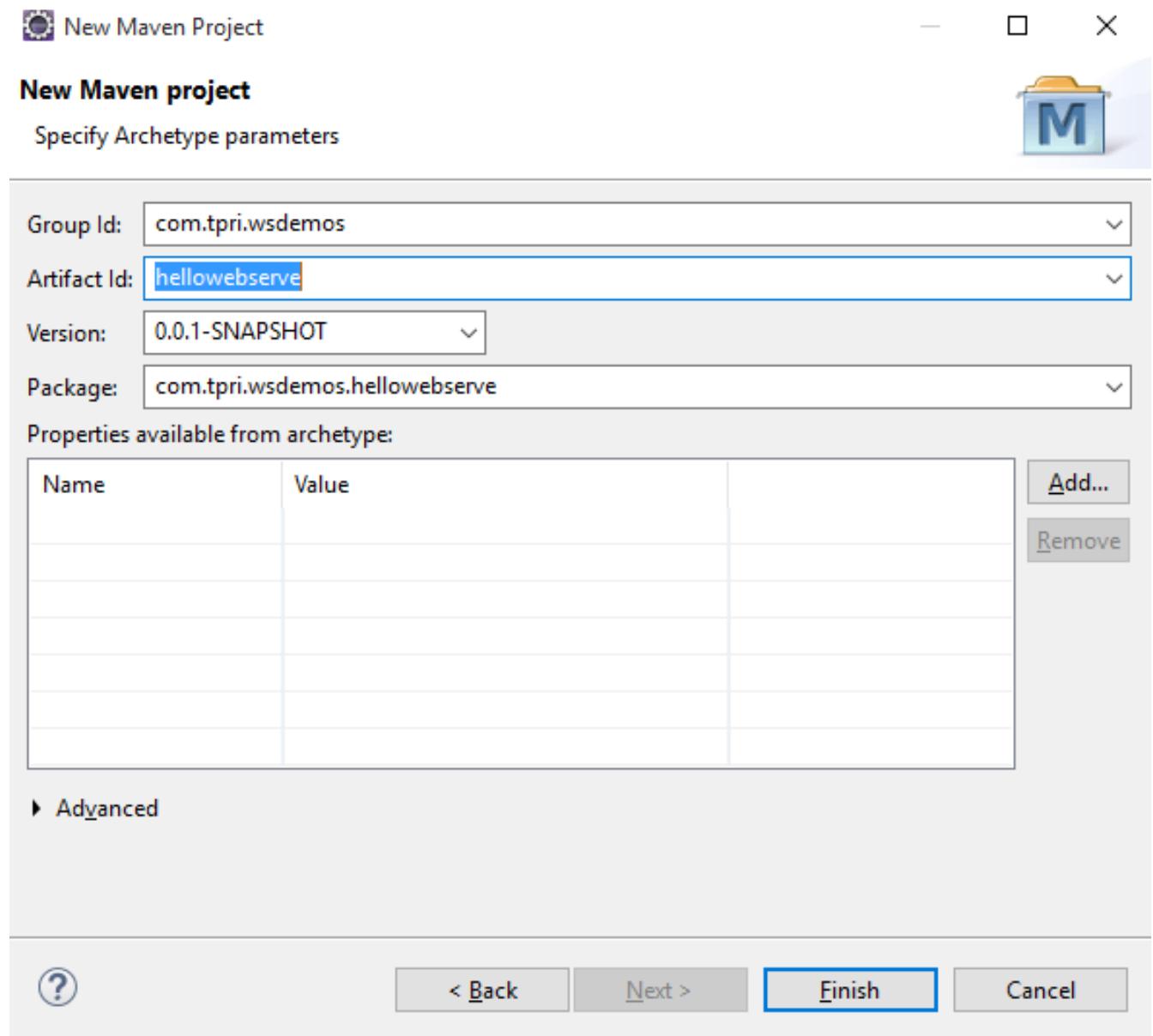
MAVEN ARCHETYPE

An archetype is a templating tool that standardizes the folder structure and artifacts of a project build.



STEP 3

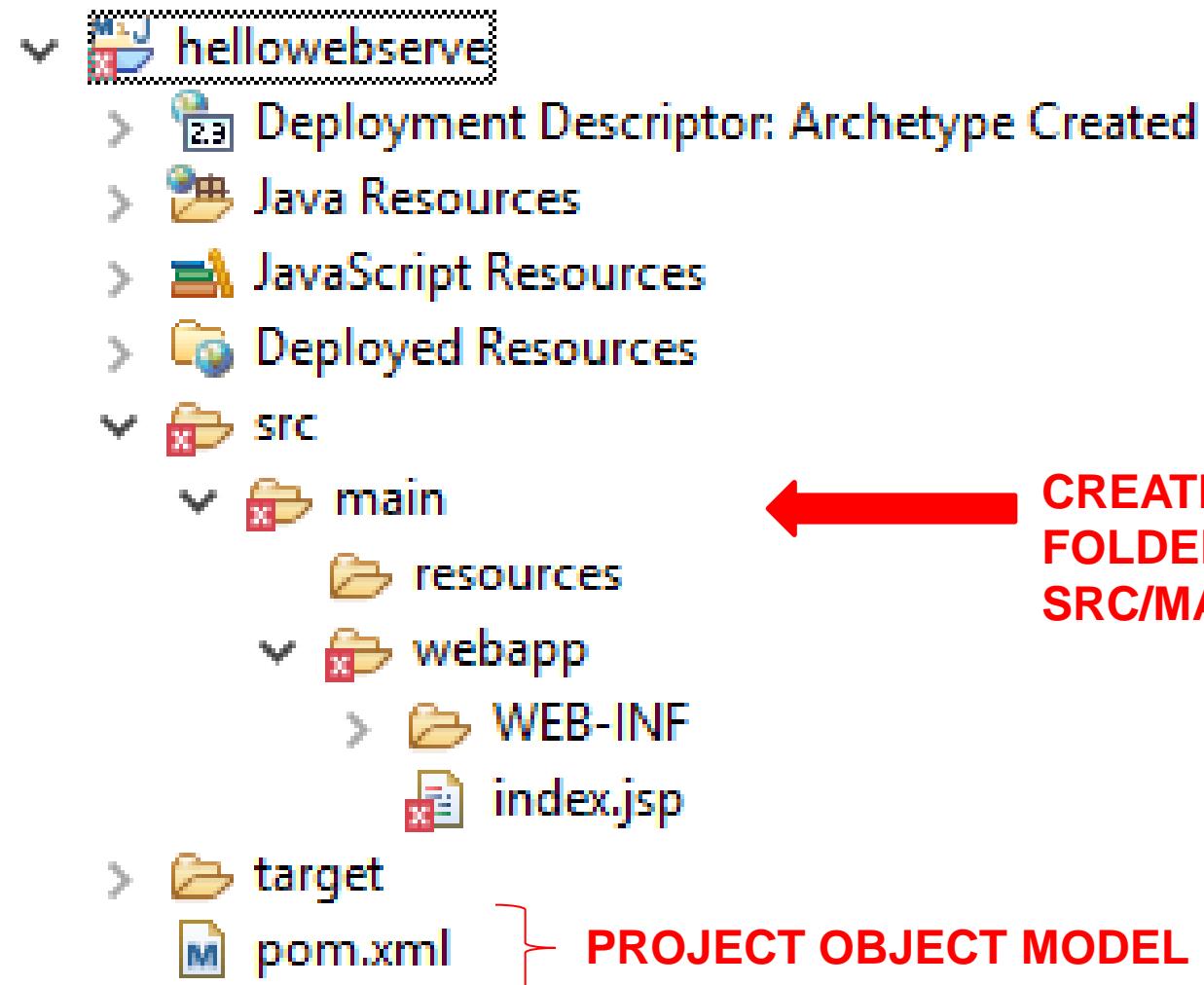
- **DEFINE GROUPID => UNIQUE PROJECT IDENTIFIER**
- **DEFINE ARTIFACTID=> “helloworld”**



Apache CXF Project Structure

STEP 4

- THE FOLLOWING FOLDER STRUCTURE IS CREATED IN ECLIPSE



CREATE A NEW
FOLDER =>
SRC/MAIN/JAVA

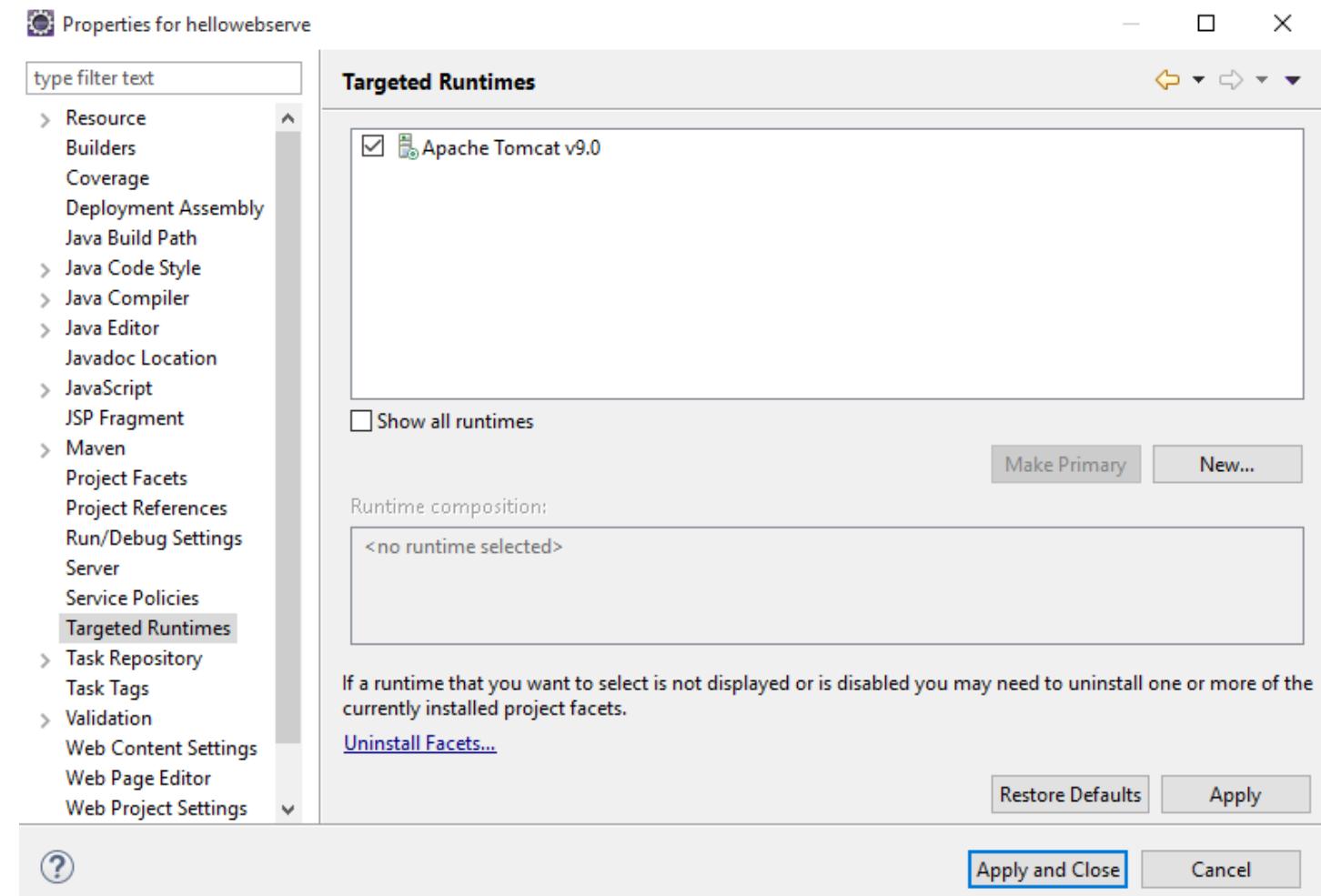
POM.XML

- Project Object Model is used
- for project configuration
 - for defining dependencies
 - for build configurations



STEP 5

- SET TARGETED RUNTIME FOR “helloworldserve” MAVEN PROJECT



STEP 6

- MODIFY THE EXISTING POM.XML (PROJECT OBJECT MODEL) BY REMOVING JUNIT PLUGINS AND ADDING NEW PLUGINS AND DEPENDENCIES

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tpri.wsdemos</groupId>
  <artifactId>hellowebserve</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>hellowebserve Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <build>
    <finalName>hellowebserve</finalName>
  </build>
</project>
```

POM.XML

- Project Object Model is used
- for project configuration
 - for defining dependencies
 - for build configurations



STEP 7

- Add **plugins**
 - **Maven-compiler-plugin**
 - **Maven-war-plugin**
- To the POM.XML FILE**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <source>1.7</source>
        <target>1.7</target>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.1</version>
      <configuration>
        <webXml>src/main/webapp/WEB-INF/web.xml</webXml>
      </configuration>
    </plugin>
  </plugins>
  <finalName>hellows</finalName>
</build>
```



IMPLEMENTING JAX-WS

- Available in the **cx-rt-frontend-jaxws** library
- Supported by a variety of transports
- Configured through Spring Application context



STEP 8

- ADD DEPENDENCIES

- Org.apache.cxf
 - cxf-rt-bindings-soap
 - cxf-rt-transports-http
 - cxf-rt-frontend-jaxws
 - cxf-rt-rs-extension-providers

- Org.springframework

- spring-core
- spring-context
- spring-web

```
<dependencies>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-bindings-soap</artifactId>
        <version>3.0.3</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-transports-http</artifactId>
        <version>3.0.3</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxws</artifactId>
        <version>3.0.3</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-rs-extension-providers</artifactId>
        <version>3.0.3</version>
        <scope>compile</scope>
    </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>
</dependencies>
```

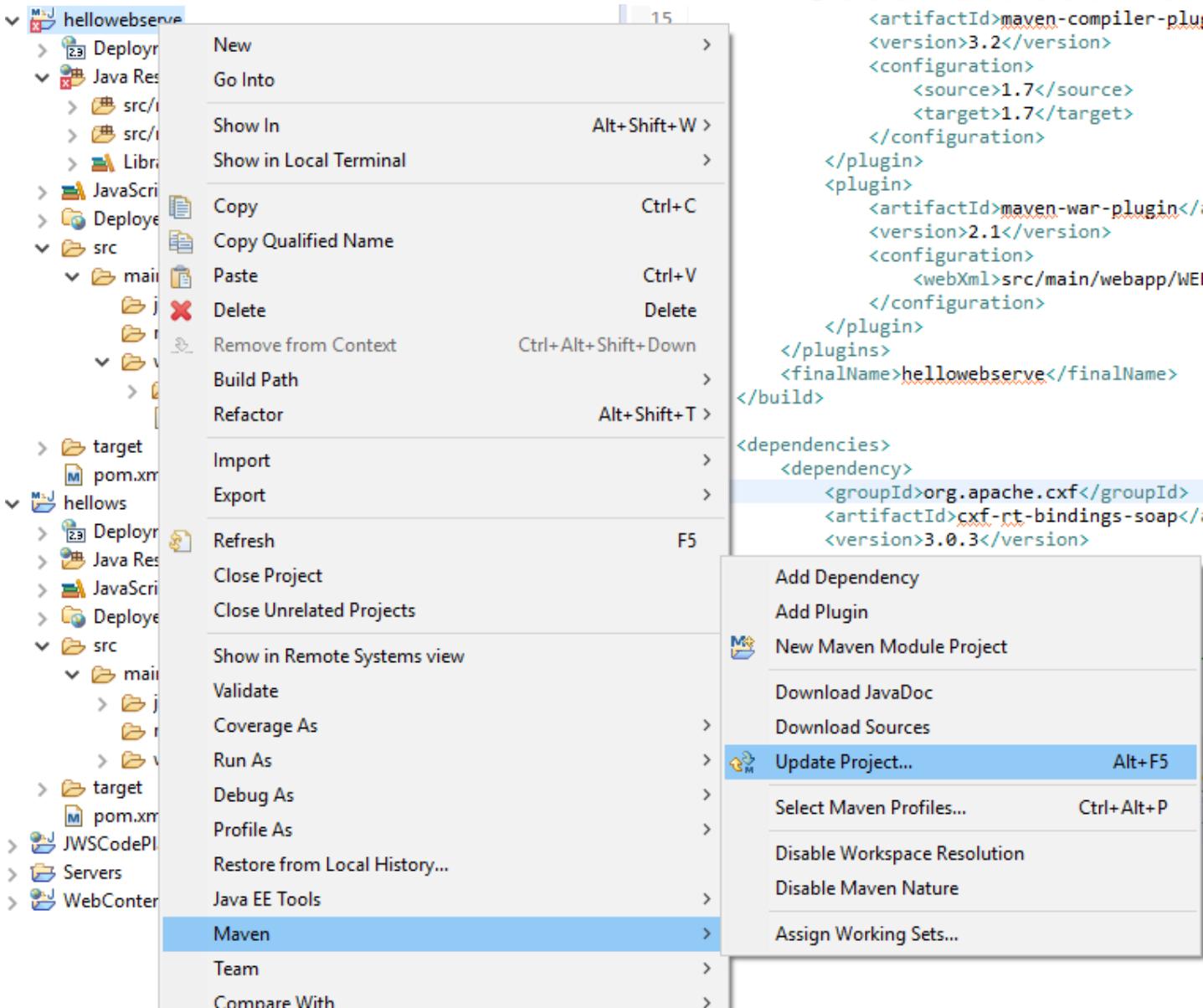
APACHE
CXF
DEPENDENCIES

SPRING
DEPENDENCIES



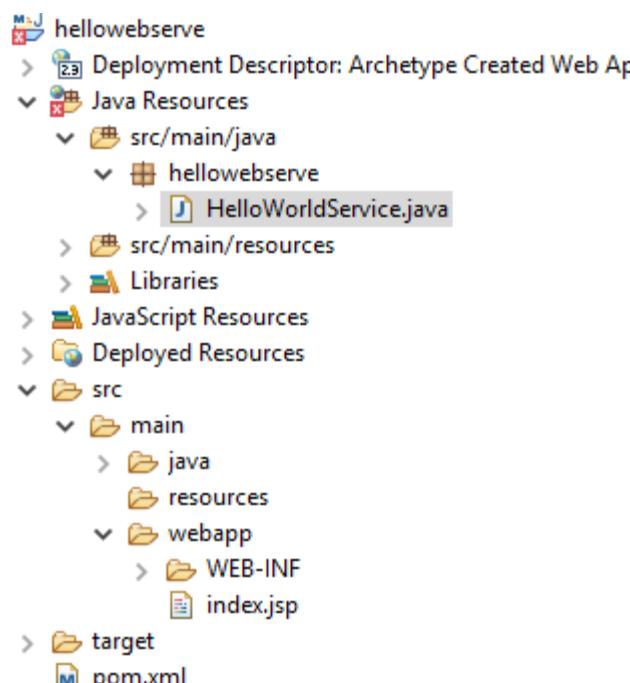
STEP 9

- POST MODIFICATION OF POM.XML PLEASE UPDATE THE MAVEN PROJECT (ALT+F5)



STEP 10

- CREATE JAVA WEB SERVICE WITH ANNOTATIONS

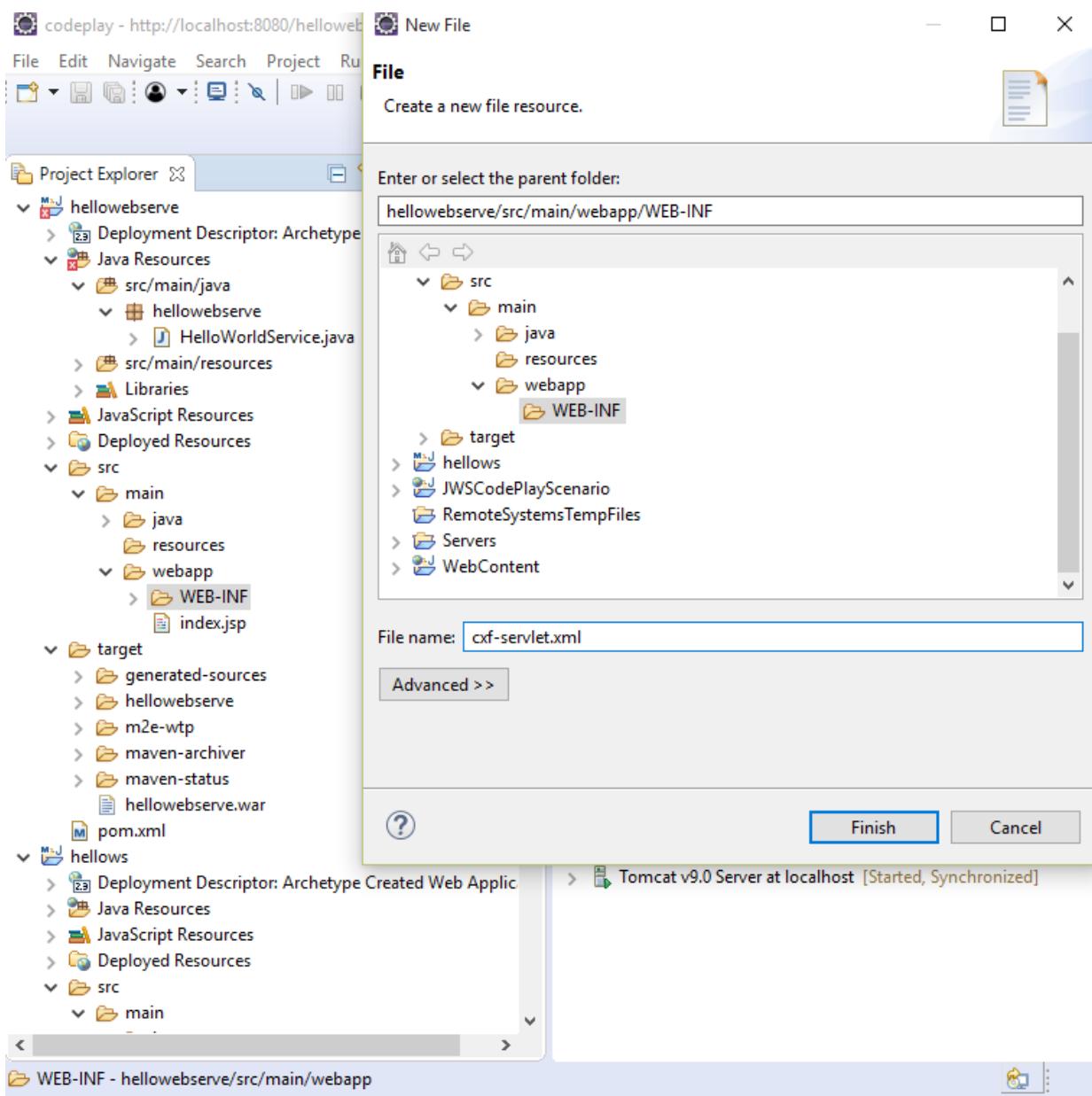
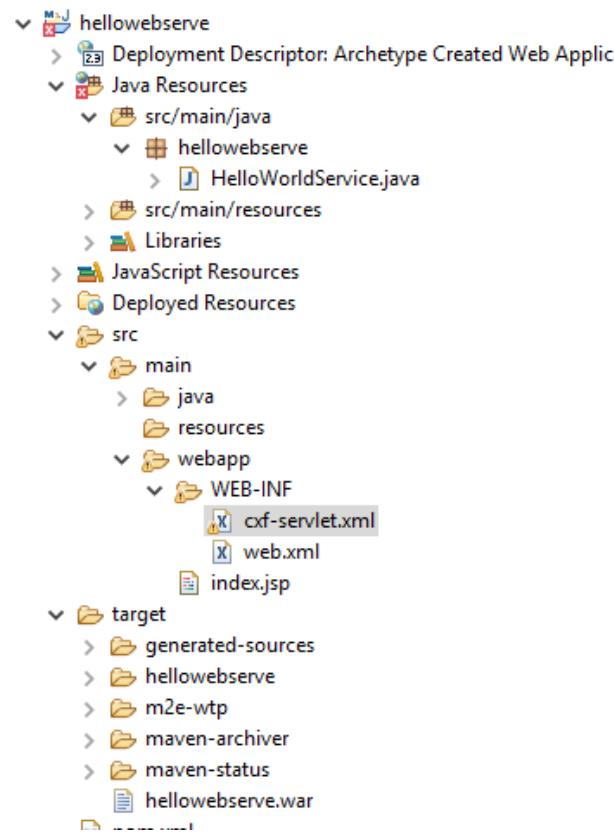


```
1 package helloworldserve;  
2  
3 import javax.jws.WebMethod;  
4 import javax.jws.WebService;  
5  
6 @WebService  
7 public class HelloWorldService {  
8  
9     @WebMethod  
10    public String callHello() {  
11        return "Hello TPRI:Syed Awase";  
12    }  
13}  
14
```



STEP 11

- CREATE A FILE `cxf-servlet.xml` in



CXF SERVLET

- Request processing for web service endpoints
- Available in the **cxf-rt-transports-http** library
- Supports the creation of Spring's Application context



STEP 12

- UPDATE THE CXF-SERVLET.XML FILE WITH BEAN CLASS INFO

cxf-servlet.xml

CXF-Servlet is used

- CXF bean import
- defining JAX-WS end point

CXF WEB SERVICES CONFIGURATION

```
<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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd">

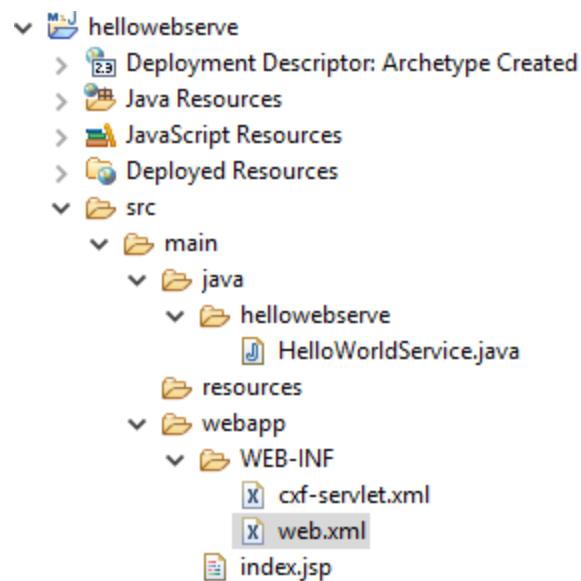
    <jaxws:server id="hellowebserve" address="/hello">
        <jaxws:serviceBean>
            <bean class="hellowebserve.HelloWorldService" />
        </jaxws:serviceBean>
    </jaxws:server>
</beans>
```



STEP 13

- DEFINE THE WEB APP CONFIGURATIONS FOR APACHE CXF AND MAPPINGS TO RENDER THE SERVICES.

- UPDATE THE WEB.XML AS SHOWN



CXF Servlet Web Descriptor

```
<!DOCTYPE web-app PUBLIC  
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
"http://java.sun.com/dtd/web-app_2_3.dtd" >  
  
<web-app>  
  <display-name>Archetype Created Web Application</display-name>  
  <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>  
</web-app>
```

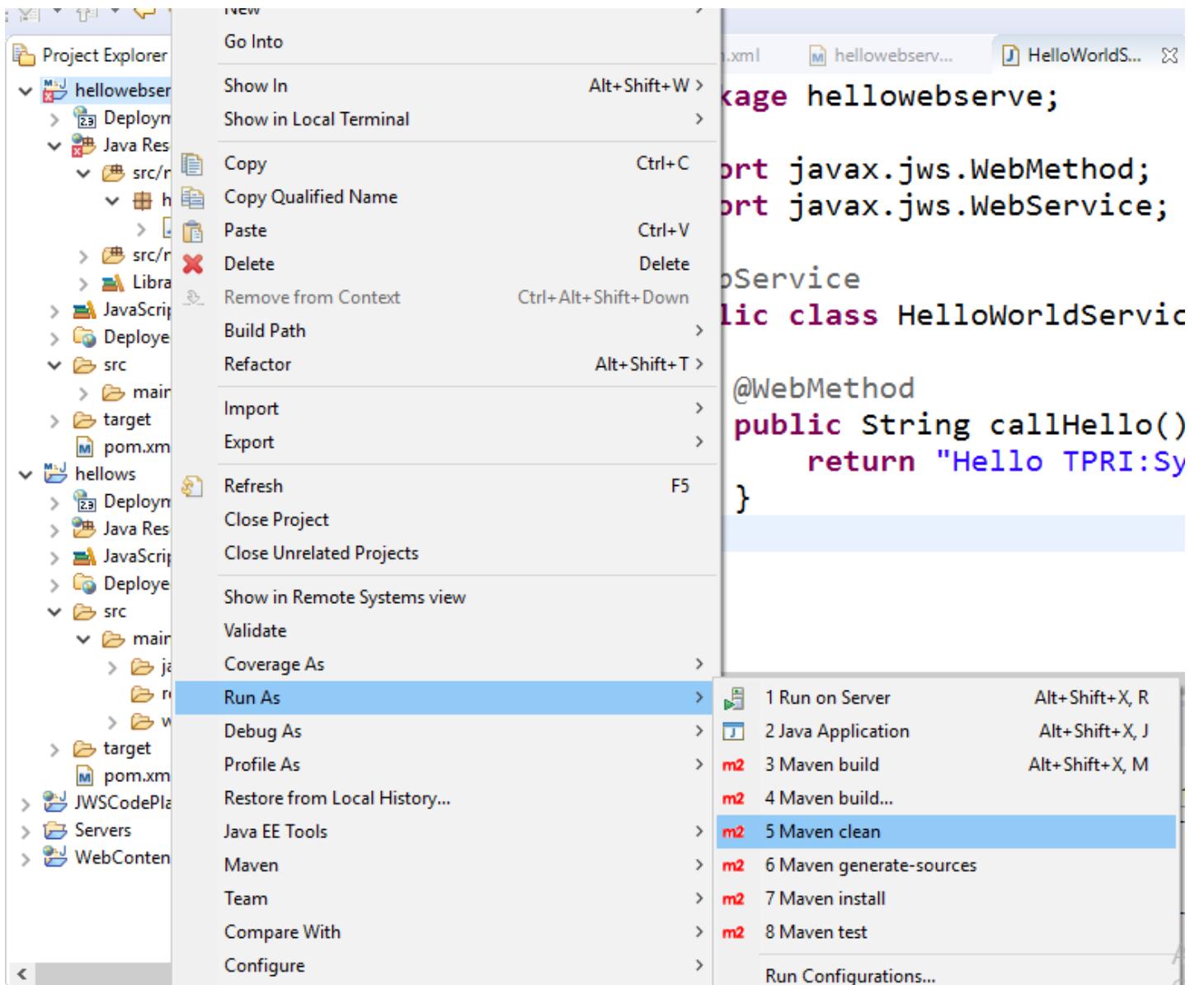
WEB.XML

- Web.XML is used**
- for Spring context
 - CXF Servlet Configuration
 - CXF Servlet mapping
 - CXF url-pattern definitions



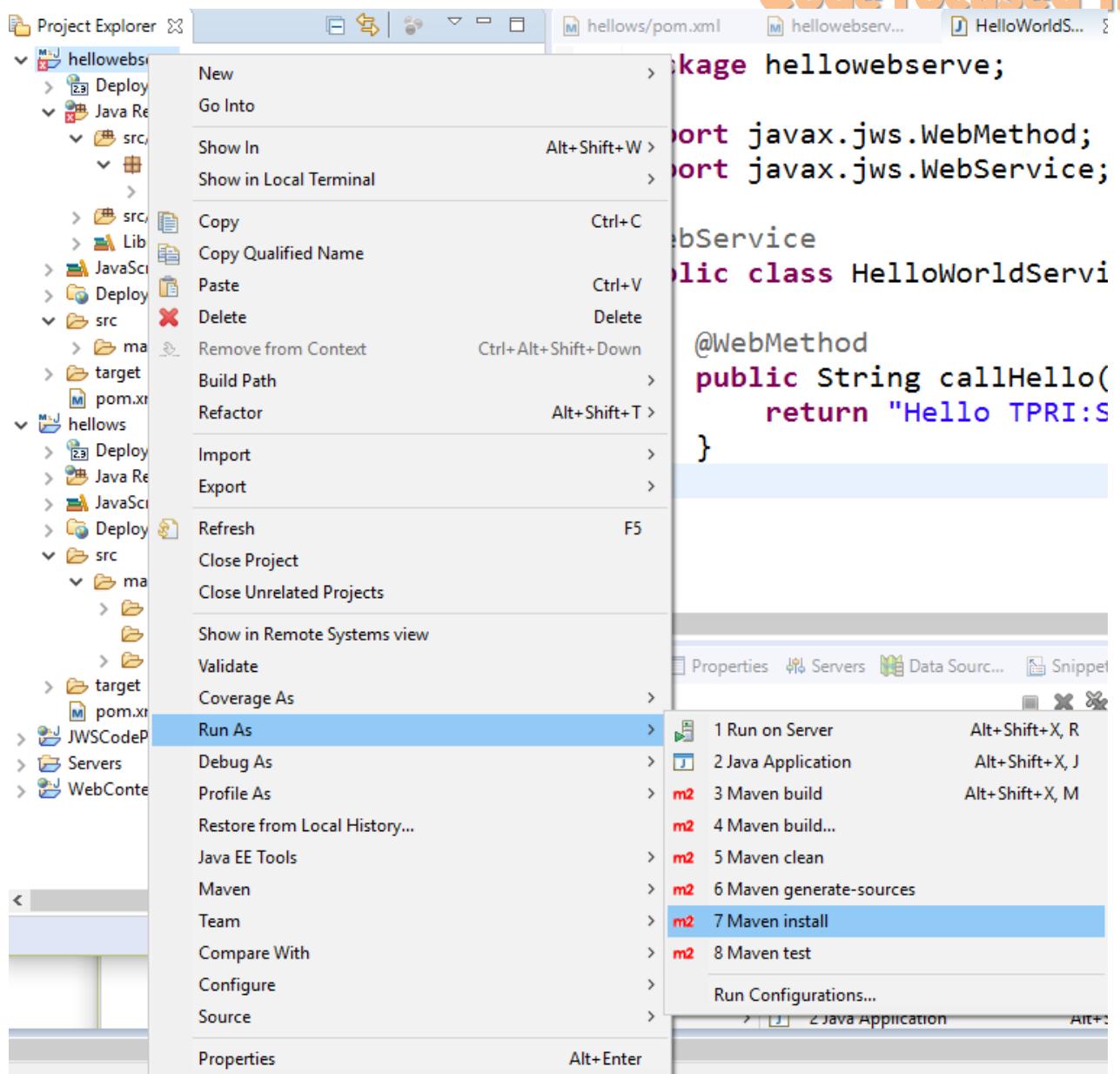
STEP 14

- MAVEN CLEAN THE APPLICATION



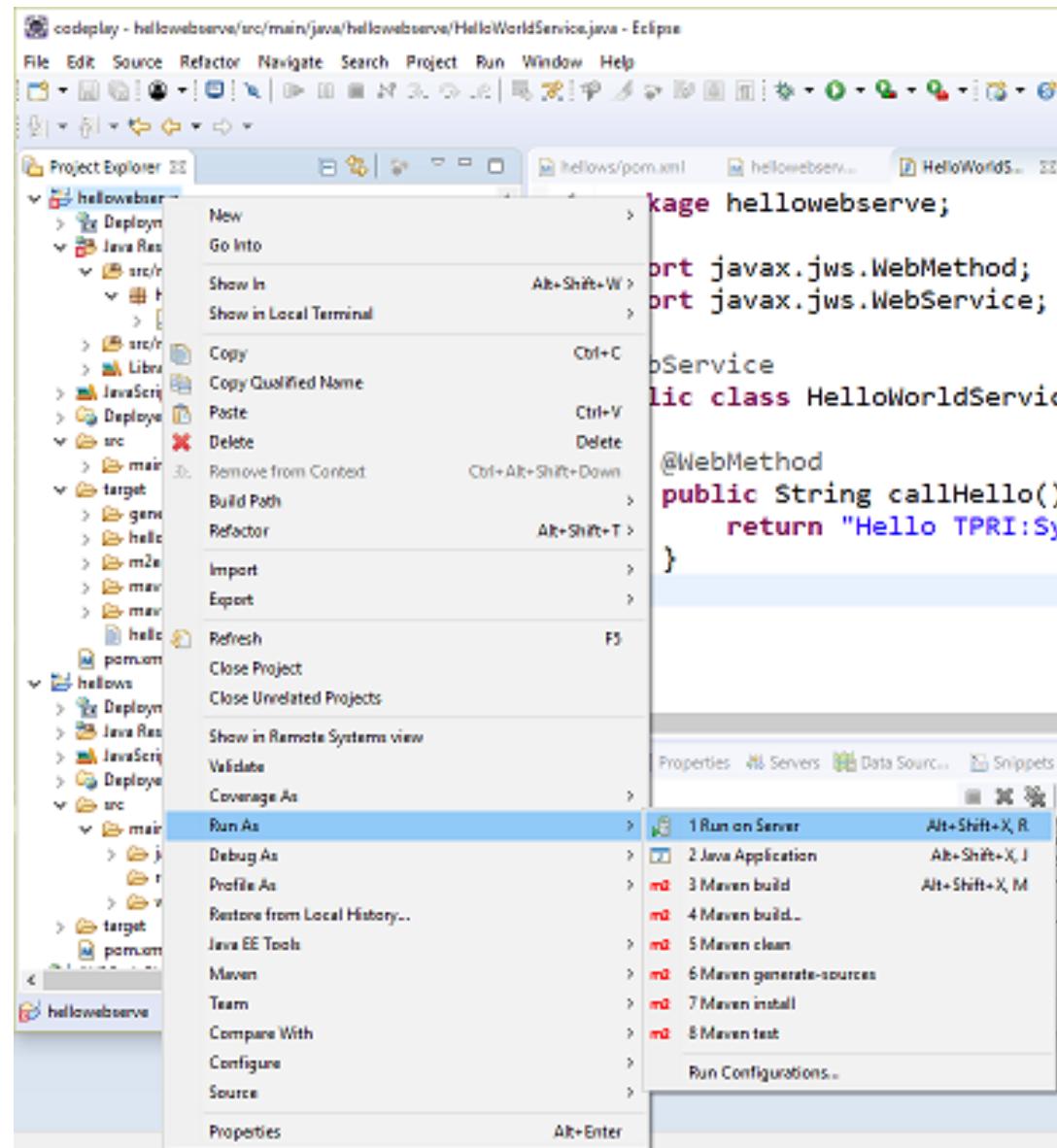
STEP 15

- MAVEN INSTALL FOR
INSTALLING THE
DEPENDENCIES OUTLINED
IN POM.XML



STEP 16

- RUN YOUR APPLICATION



STEP 17

- navigate to

<http://localhost:8080/hellowebserve/>

- Navigate to the web services @

<http://localhost:8080/hellowebserve/services/>

- outcome=> a simple web service and corresponding wsdl file!

CXF - Service list

localhost:8080/hellowebserve/services

Available SOAP services:

HelloWorldService	Endpoint address: http://localhost:8080/hellowebserve/services/hello WSDL : http://hellowebserve/HelloWorldServiceService Target namespace: http://hellowebserve/
-------------------	--

Available RESTful services:

localhost:8080/hellowebserve/services/hello?wsdl

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

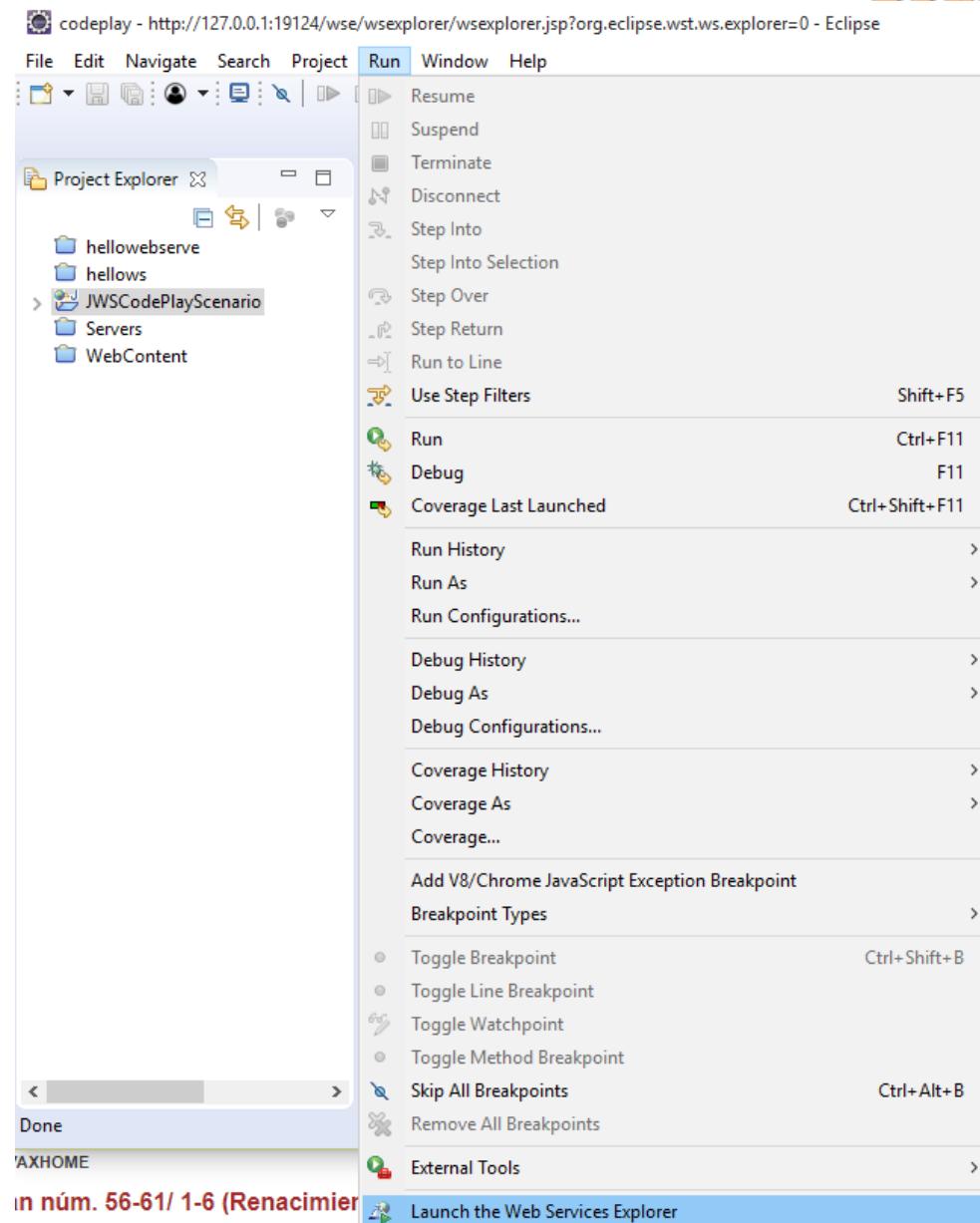
```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://hellowebserve/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://hellowebserve/" elementFormDefault="unqualified" targetNamespace="http://hellowebserve/" version="1.0">
    <wsdl:types>
      <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns:tnsc="http://hellowebserve/" elementFormDefault="unqualified" targetNamespace="http://hellowebserve/">
        <xss:element name="callHello" type="tnsc:callHello"/>
        <xss:element name="callHelloResponse" type="tnsc:callHelloResponse"/>
        <xss:complexType name="callHello">
          <xss:sequence>
            <xss:sequence/>
          </xss:sequence>
        </xss:complexType>
        <xss:complexType name="callHelloResponse">
          <xss:sequence>
            <xss:element minOccurs="0" name="return" type="xs:string"/>
          </xss:sequence>
        </xss:complexType>
      </xsschema>
    </wsdl:types>
    <wsdl:message name="callHello">
      <wsdl:part element="tns:callHello" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="callHelloResponse">
      <wsdl:part element="tns:callHelloResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:portType name="HelloWorldService">
      <wsdl:operation name="callHello">
        <wsdl:input message="tns:callHello" name="callHello"/>
        <wsdl:output message="tns:callHelloResponse" name="callHelloResponse"/>
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="HelloWorldServiceSoapBinding" type="tns:HelloWorldService">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="callHello">
        <soap:operation soapAction="" style="document"/>
        <wsdl:input name="callHello">
          <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="callHelloResponse">
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="HelloWorldServiceService">
      <wsdl:port binding="tns:HelloWorldServiceSoapBinding" name="HelloWorldServicePort">
        <soap:address location="http://localhost:8080/hellowebserve/services/hello"/>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Activate Win
Go to Settings to



STEP 18

- LAUNCH THE WEB SERVICES EXPLORER

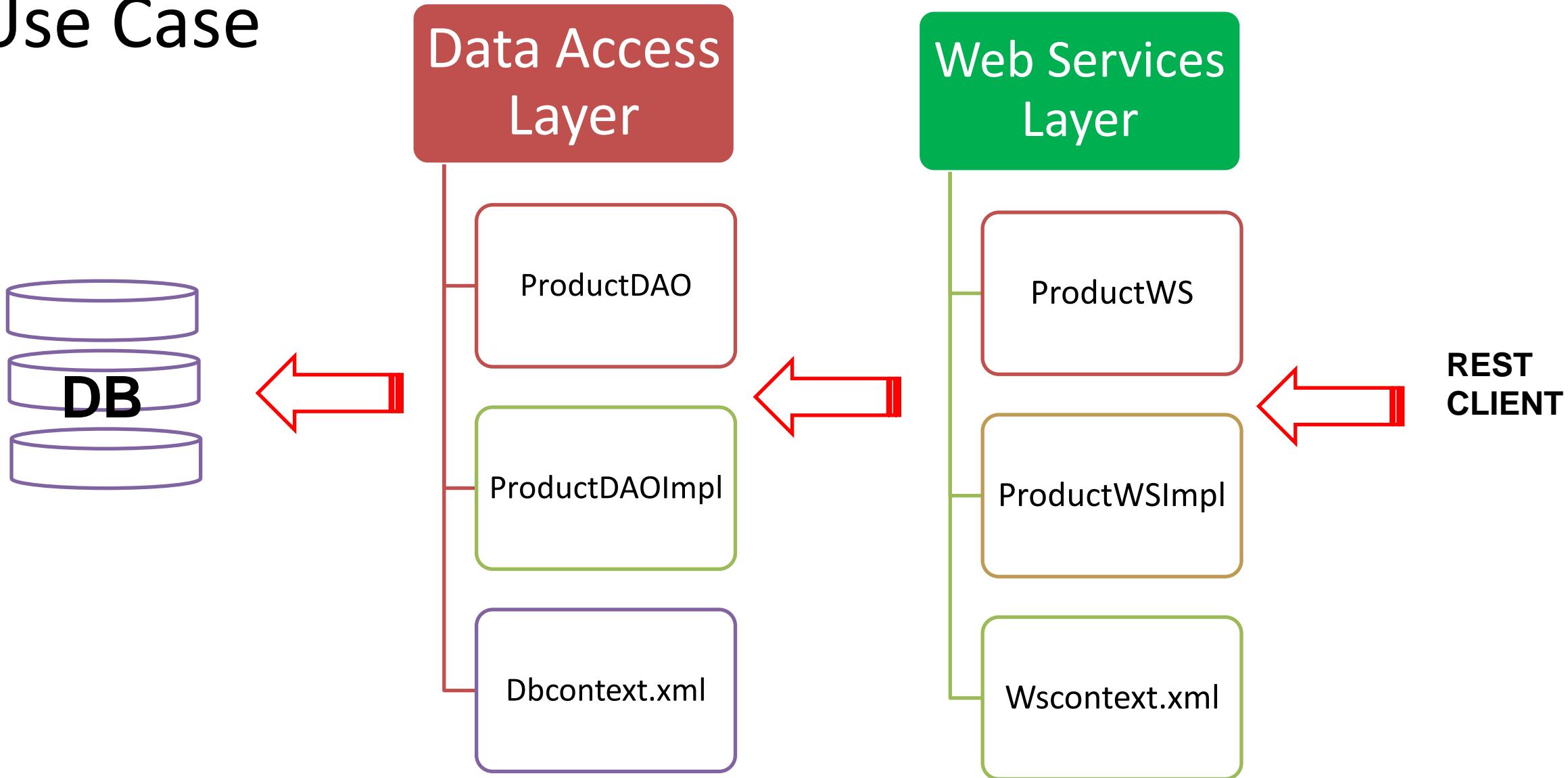




SYED AWASE

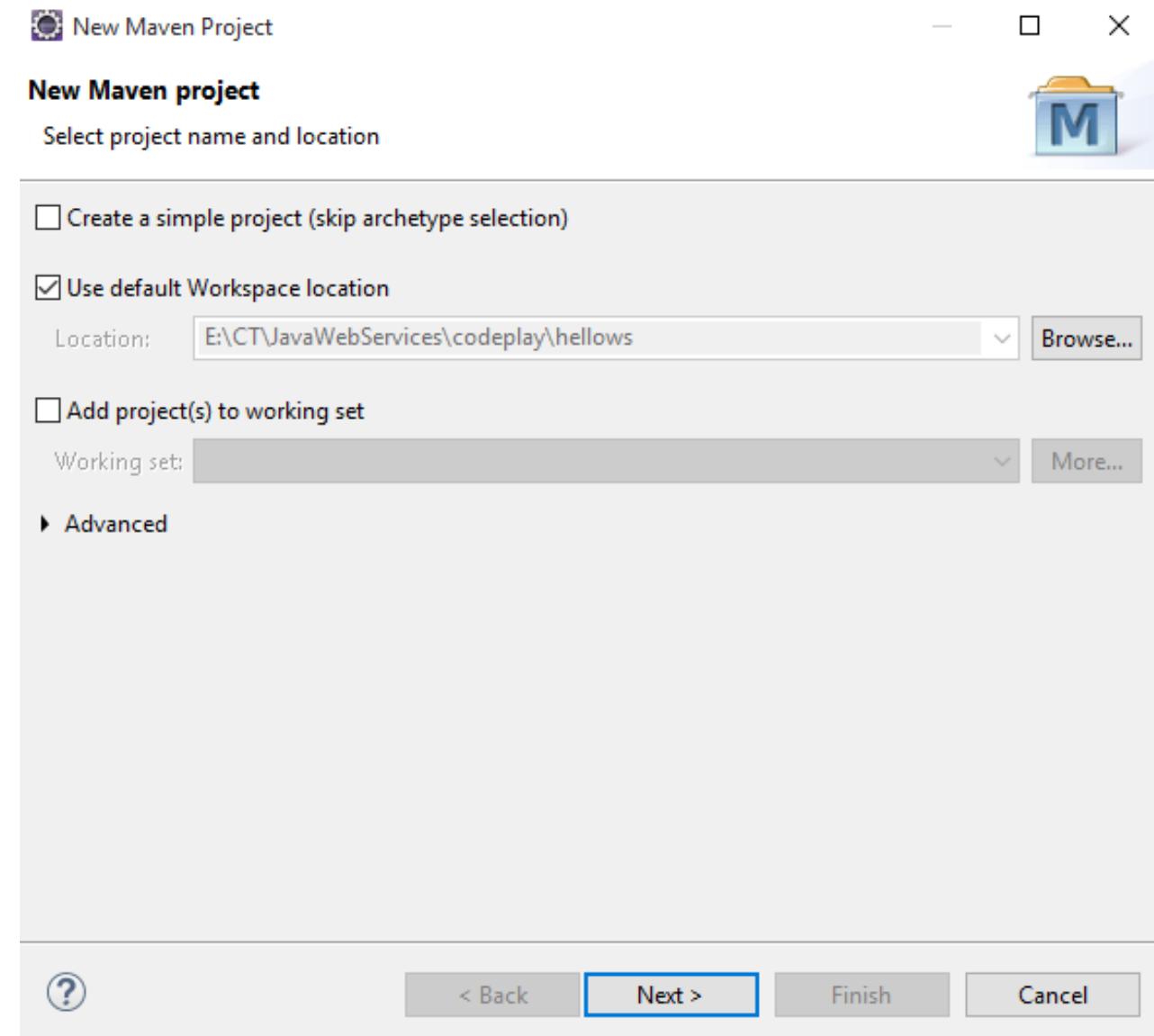
VIII(B): : JAVA WEB SERVICES: CRUD DEMO WITH APACHE CXF

Use Case



STEP 1

- **CREATE A NEW MAVEN PROJECT USING DEFAULT WORKSPACE LOCATION.**

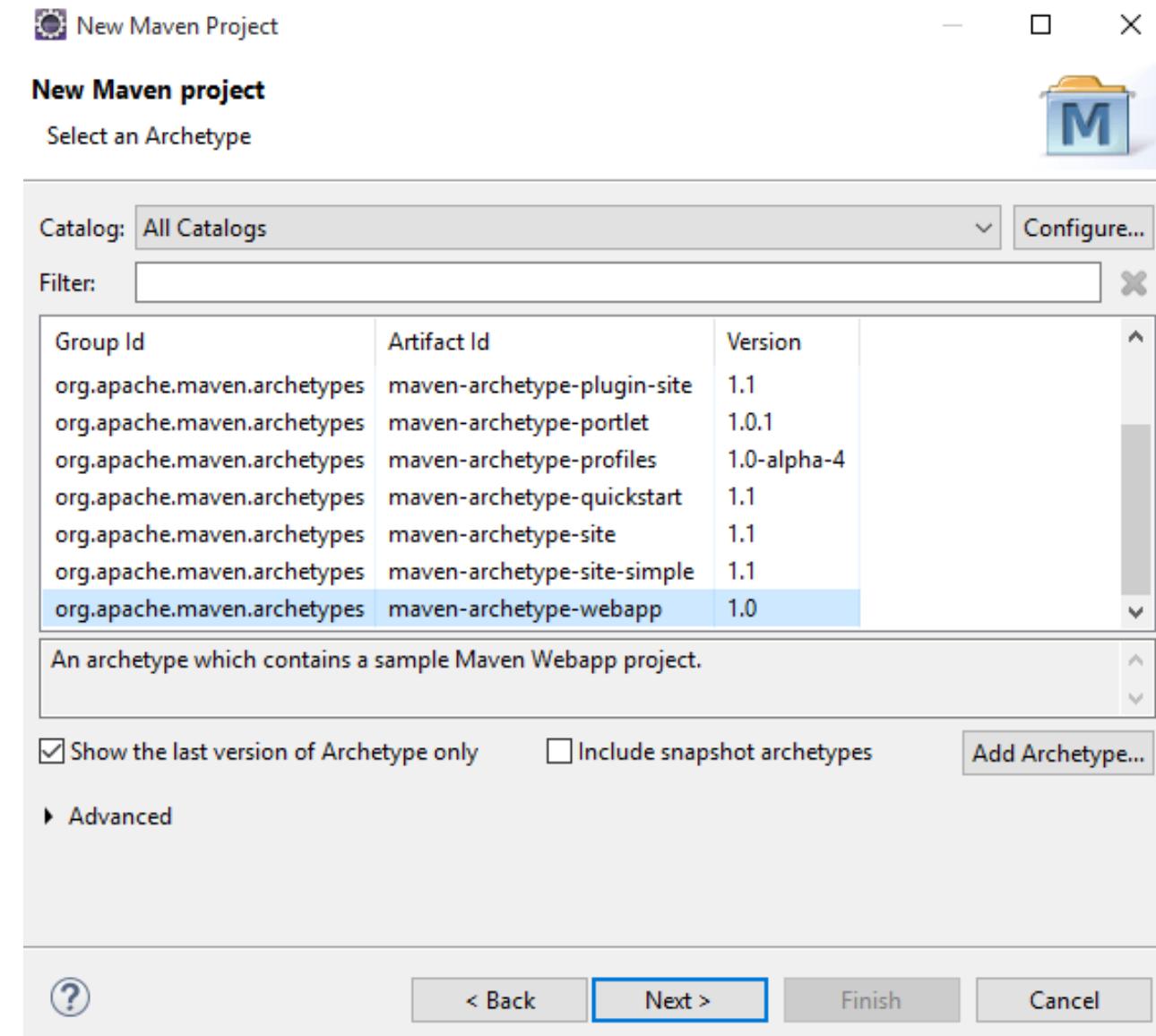


STEP 2

- SELECT
ORG.APACHE.MAVEN.
ARCHETYPES =>
**MAVEN.ARCHETYPE-
WEBAPP**

MAVEN ARCHETYPE

An archetype is a templating tool that standardizes the folder structure and artifacts of a project build.



Step 3

- Add dependencies to pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxws</artifactId>
        <version>3.0.3</version>
    </dependency>
    <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-rs-service-description</artifactId>
        <version>3.0.0-milestone1</version>
    </dependency>
</dependencies>
```



Step 4

- add spring-core and jdbc dependencies to pom.xml

```
<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.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
```



Step 5

- add jackson-jaxrs and mysql-connector-java to pom.xml

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-jaxrs</artifactId>
    <version>1.9.13</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
</dependency>
```

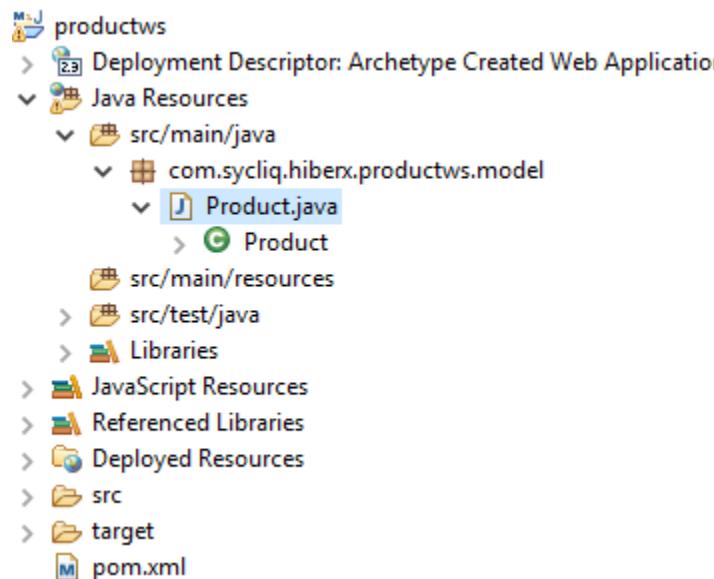
JSON Support

Database
Connector



Step 6

- create a POJO product model



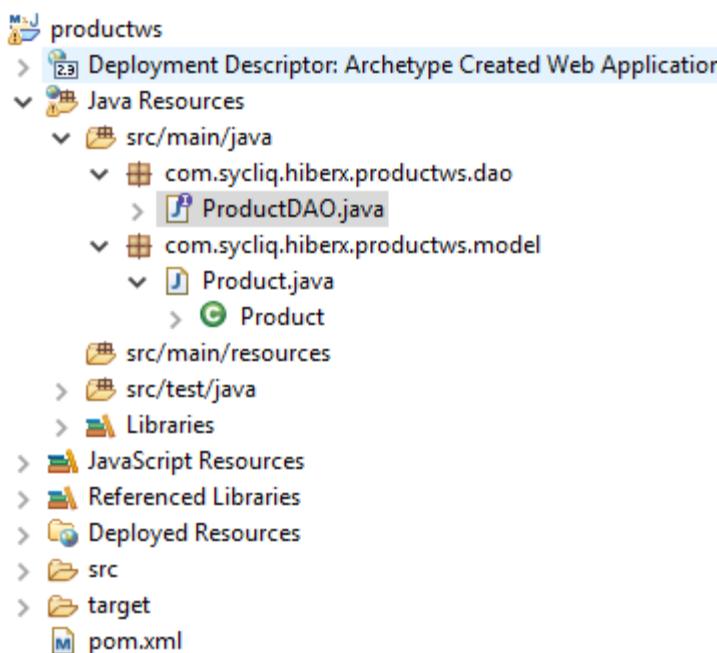
```
package com.sycliq.hiberx.productws.model;

public class Product {
    private int id;
    private String name;
    private String description;
    private int price;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }
}
```



Step 7

- create ProductDAO interface

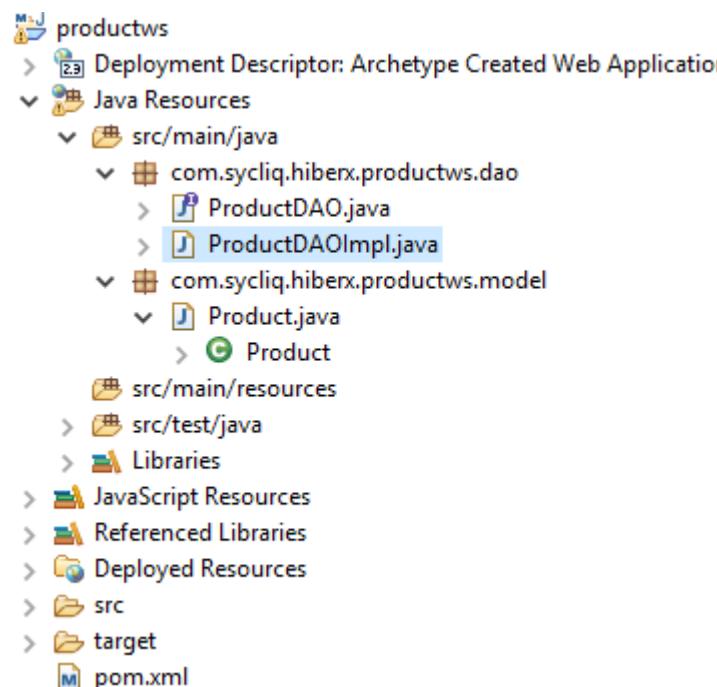


```
package com.sycliq.hiberx.productws.dao;  
  
import com.sycliq.hiberx.productws.model.Product;  
  
public interface ProductDAO {  
  
    void create(Product product);  
    void update(int id, int price);  
    void delete(int id);  
    Product find(int id);  
}
```



Step 8

- create ProductDAOImpl class to implement the ProductDAO



```
package com.sycliq.hiberx.productws.dao;

import com.sycliq.hiberx.productws.model.Product;

public class ProductDAOImpl implements ProductDAO{

    public void create(Product product) {
        // TODO Auto-generated method stub
    }

    public void update(int id, int price) {
        // TODO Auto-generated method stub
    }

    public void delete(int id) {
        // TODO Auto-generated method stub
    }

    public Product find(int id) {
        // TODO Auto-generated method stub
        return null;
    }
}
```



Step 9

- Create ProductDAOImpl
implements ProductDAO

```
package com.sycliq.hiberx.productws.dao;

import org.springframework.jdbc.core.JdbcTemplate;

import com.sycliq.hiberx.productws.dao.mappers.ProductRowMapper;
import com.sycliq.hiberx.productws.model.Product;

public class ProductDAOImpl implements ProductDAO{

    private JdbcTemplate jdbcTemplate;

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void create(Product product) {
        // TODO Auto-generated method stub
        jdbcTemplate.update("insert into product
value("+product.getId()+","+product.getName()+",'"+product.getDescription()+"','"+product.getPrice()+"')");

    }

    public void update(int id, int price) {
        // TODO Auto-generated method stub
        jdbcTemplate.update("update product set price="+price+"where id="+id);

    }

    public void delete(int id) {
        // TODO Auto-generated method stub
        jdbcTemplate.update("delete from product where id="+id);

    }

    public Product find(int id) {
        // TODO Auto-generated method stub
        Product product =jdbcTemplate.queryForObject("select * from product where id="+id, new ProductRowMapper());
        return product;
    }

}
```

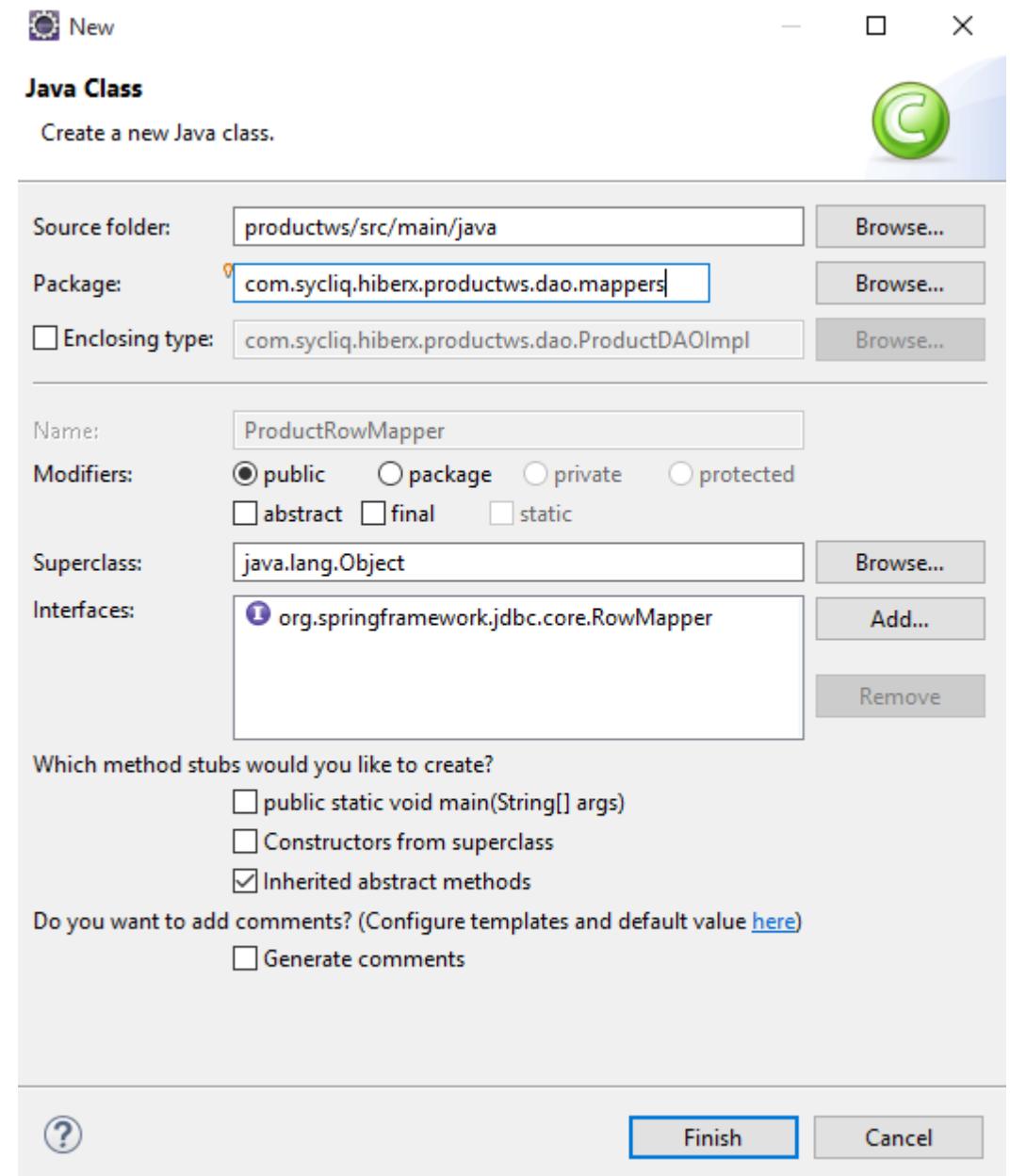


Step 10

- create
ProductRowMapper

```
public Product find(int id) {  
    // TODO Auto-generated method stub  
    jdbcTemplate.queryForObject("select * from product where id=?"+id, new ProductRowMapper());  
    return null;  
}
```

ProductRowMapper cannot be resolved to a type
4 quick fix available:
① Create class 'ProductRowMapper'
② Change to 'RowMapper' (org.springframework.jdbc.core)
③ Change to 'Product' (com.oracle.jctk.jt)
④ Fix project setup...



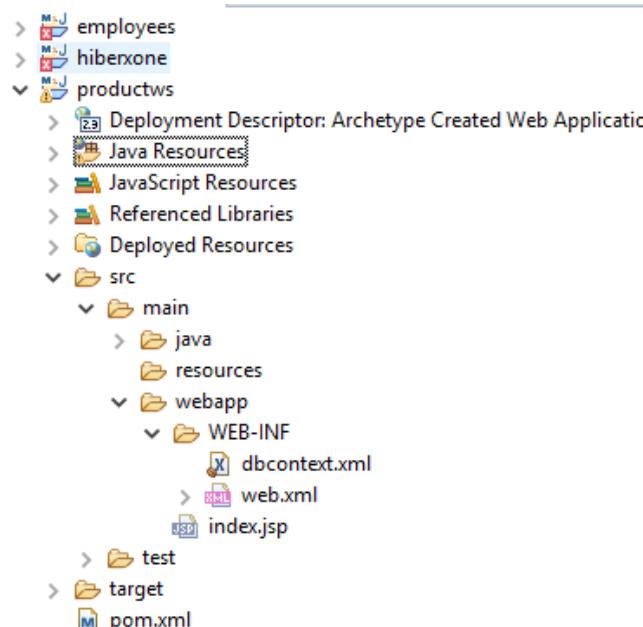
Step 11

```
package com.sycliq.hiberx.productws.dao.mappers;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import org.springframework.jdbc.core.RowMapper;  
import com.sycliq.hiberx.productws.model.Product;  
  
public class ProductRowMapper implements RowMapper<Product> {  
  
    public Product mapRow(ResultSet rs, int rowNum) throws SQLException {  
        // TODO Auto-generated method stub  
        Product product = new Product();  
        product.setId(rs.getInt(1));//type integer  
        product.setName(rs.getString(2));//type String  
        product.setDescription(rs.getString(3));//type String  
        product.setPrice(rs.getInt(4));//type integer  
        return product;  
    }  
}
```



Step 12

- create dbcontext.xml file to wire the DAO with spring framework.



```
<?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:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- bean database configuration -->
    <bean id="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/test" />
        <property name="username" value="mysqlusr" />
        <property name="password" value="user123" />
    </bean>

    <!-- jdbcTemplate configuration -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>

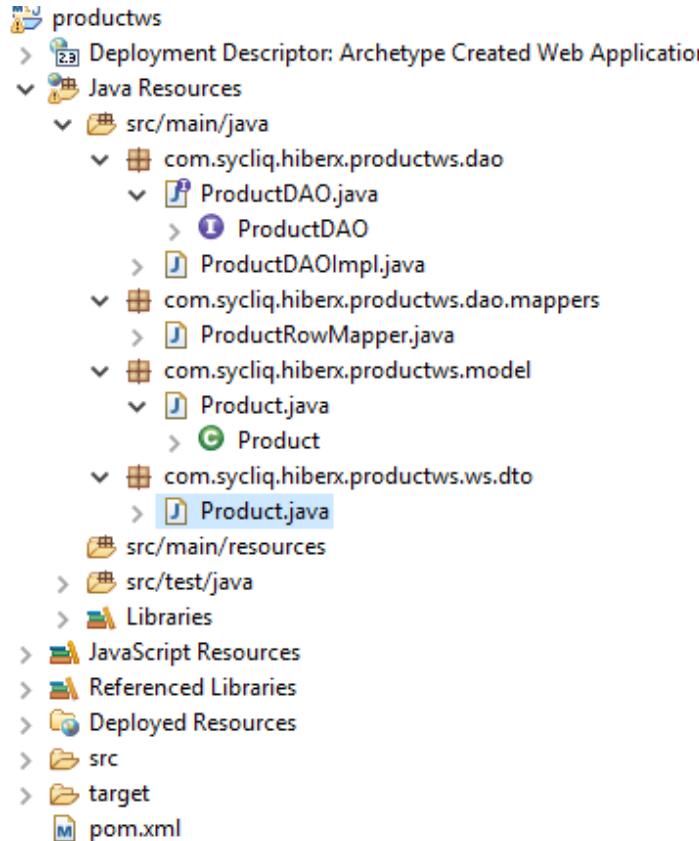
    <!-- configure the DAO to inject the DAO in the webservice -->
    <bean id="ProductDAO" class="com.sycliq.hiberx.productws.dao.ProductDAOImpl">
        <property name="jdbcTemplate" ref="jdbcTemplate" />
    </bean>

</beans>
```



Step 13

- create Product Model in DTO layer



```
package com.sycliq.hiberx.productws.ws.dto;

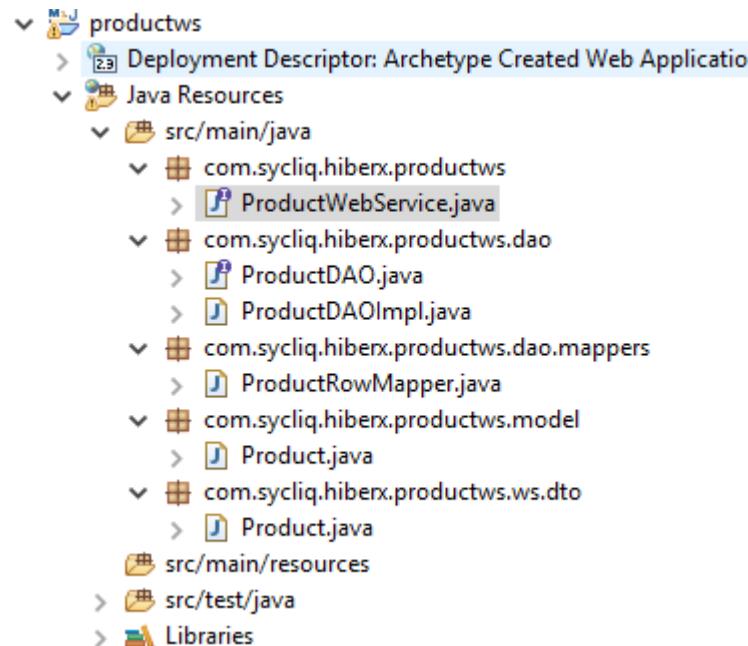
import javax.xml.bind.annotation.XmlRootElement;

/**
 * @author SyedAwase
 * We are creating a Product Model for Web Service Layer
 * to keep it decoupled with the entity layer
 */
@XmlRootElement
public class Product {
    private int id;
    private String name;
    private String description;
    private int price;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }
}
```



Step 14

- create the web service interface for product



```
package com.sycliq.hiberx.productws;

import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

import com.sycliq.hiberx.productws.ws.dto.Product;

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

    @POST
    @Path("/products")
    Response create(Product product);

    @PUT
    @Path("/products")
    Response update(Product product);

    @DELETE
    @Path("/products/{id}")
    Response delete(@PathParam("id") int id);

    @GET
    @Path("/products/{id}")
    Product find(@PathParam("id") int id);

}
```



Step 15

- create ProductWebServiceImpl implementation of the ProductWebService interface

```
package com.sycliq.hiberx.productws;
import javax.ws.rs.core.Response;
import com.sycliq.hiberx.productws.dao.ProductDAO;
import com.sycliq.hiberx.productws.ws.dto.Product;

public class ProductWebServiceImpl implements ProductWebService {

    private ProductDAO dao;
    public ProductDAO getDao() { return dao; }
    public void setDao(ProductDAO dao) { this.dao = dao; }

    public Response create(Product product) {
        // TODO Auto-generated method stub
        //create an instance of Product Entity
        com.sycliq.hiberx.productws.model.Product productEntity = new com.sycliq.hiberx.productws.model.Product();
        productEntity.setId(product.getId());
        productEntity.setDescription(product.getName());
        productEntity.setPrice(product.getPrice());
        dao.create(productEntity);
        return Response.ok().build();
    }

    public Response update(Product product) {
        // TODO Auto-generated method stub
        dao.update(product.getId(), product.getPrice());
        return Response.ok().build();
    }

    public Response delete(int id) {
        // TODO Auto-generated method stub
        dao.delete(id);
        return Response.ok().build();
    }

    public Product find(int id) {
        // TODO Auto-generated method stub
        com.sycliq.hiberx.productws.model.Product productEntity = dao.find(id);
        Product product = new Product();
        product.setId(productEntity.getId());
        product.setName(productEntity.getName());
        product.setDescription(productEntity.getDescription());
        product.setPrice(productEntity.getPrice());
        return product;
    }
}
```



Step 16

- create spring configuration for restful web services => wscontext.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/soap
                           http://cxf.apache.org/schemas/configuration/soap.xsd
                           http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
                           http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd">

    <bean id="serviceBean" class="com.sycliq.hiberx.productws.ProductWebServiceImpl">
        <!-- as defined in the DBCONTEXT -->
        <property name="dao" ref="ProductDAO"></property>
    </bean>

    <!-- CXF spring configurations -->
    <jaxrs:server id="productService" address="/productservice">
        <jaxrs:serviceBeans>
            <ref bean="serviceBean"/>
        </jaxrs:serviceBeans>
    </jaxrs:server>
</beans>
```



Step 17

- configuring the deployment descriptor file web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">
  <display-name>cxsf</display-name>
  <servlet>
    <description>Apache CXF Endpoint</description>
    <display-name>cxsf</display-name>
    <servlet-name>cxsf</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>cxsf</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      WEB-INF/wscontext.xml,
      WEB-INF/dbcontext.xml,
    </param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
</web-app>
```



Step 18

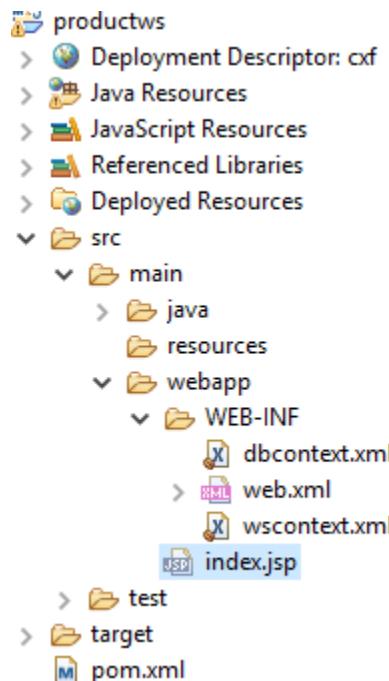
- Create mysql table for product

```
CREATE TABLE `test`.`product` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `description` VARCHAR(45) NULL,
  `price` INT NULL,
  PRIMARY KEY (`id`));
```



Step 19

- create index.jsp file

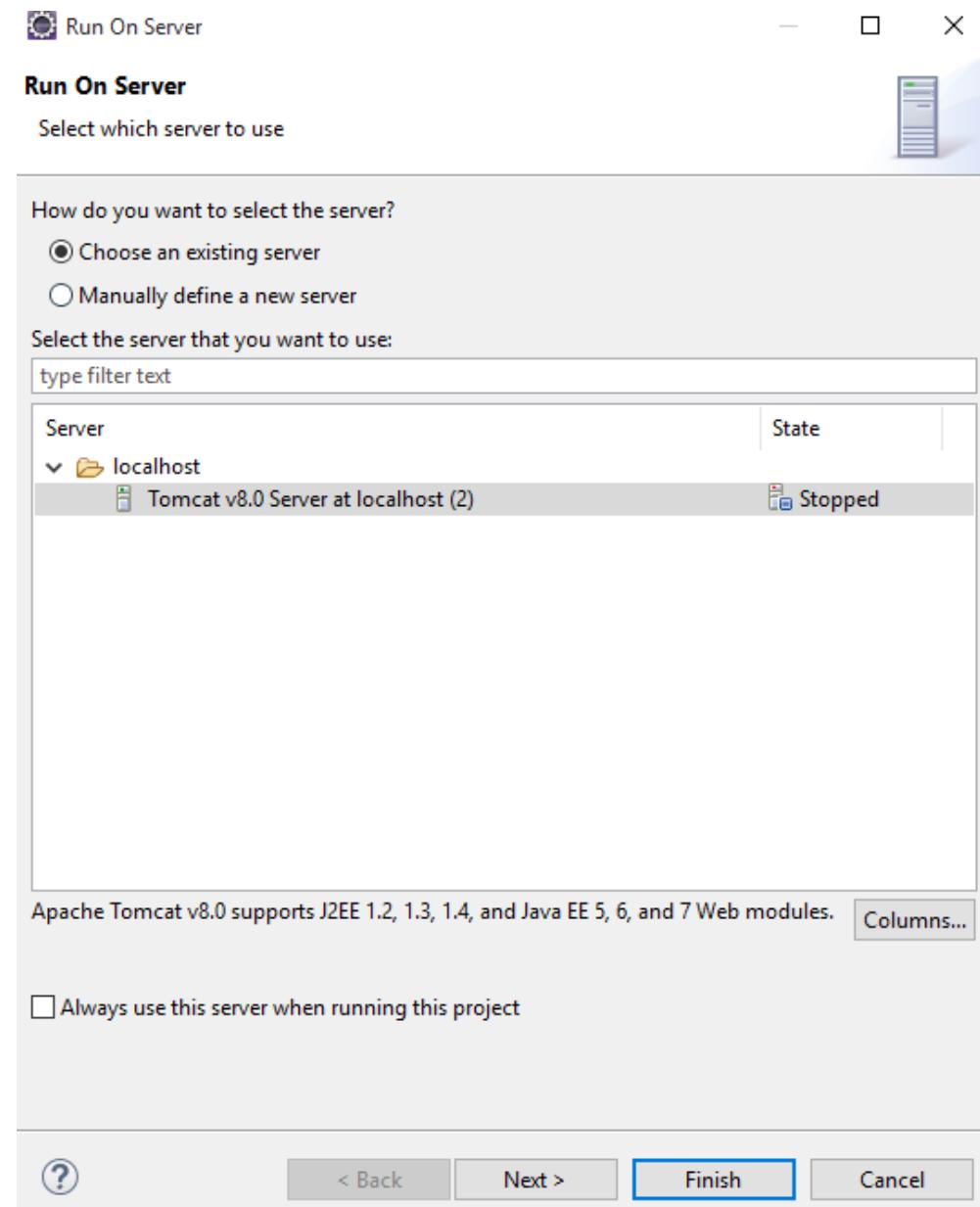


```
<html>
<body>
<h2>Welcome to WebService :<a href="services">Available Services</a></h2>
</body>
</html>
```



Step 20

- Run on Tomcat server



Step 21

-

<http://localhost:8000/productws/>



Welcome to WebService :[Available Services](#)

<http://localhost:8080/productws/services>



Available SOAP services:

Available RESTful services:

Endpoint address: <http://localhost:8080/productws/services/productservice>
WADL : http://localhost:8080/productws/services/productservice?_wadl



Step 22

- WADL generated for productservice
 - web application description language

http://localhost:8080/productws/services/productservice?_wadl

```
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"  
attributeFormDefault="unqualified">  
    <xs:element type="product" name="product"/>  
    - <xs:complexType name="product">  
        - <xs:sequence>  
            <xs:element type="xs:string" name="description" minOccurs="0"/>  
            <xs:element type="xs:int" name="id"/>  
            <xs:element type="xs:string" name="name" minOccurs="0"/>  
            <xs:element type="xs:int" name="price"/>  
        </xs:sequence>  
    </xs:complexType>  
 </xs:schema>  
</grammars>  
- <resources base="http://localhost:8080/productws/services/productservice">  
    - <resource path="/">  
        - <resource path="products">  
            - <method name="POST">  
                - <request>  
                    <representation mediaType="application/octet-stream"/>  
                </request>  
                - <response>  
                    <representation mediaType="application/xml"/>  
                    <representation mediaType="application/json"/>  
                </response>  
            </method>  
            - <method name="PUT">  
                - <request>  
                    <representation mediaType="application/octet-stream"/>  
                </request>  
                - <response>  
                    <representation mediaType="application/xml"/>  
                </response>  
            </method>  
        </resource>  
    </resource>  
</resources>
```



Step 23

- Read the webservices using postman restclient

The screenshot shows the Postman RestClient interface. At the top, there are two tabs for 'http://localhost:8080/' with orange dots, followed by a '+' button and an '...' button. Below this is a search bar with 'GET' and a dropdown menu showing 'http://localhost:8080/productws/services/productservice/products/1'. Underneath the search bar are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests', with 'Authorization' being the active tab. A dropdown for 'Type' is set to 'No Auth'. Below these tabs are four more tabs: 'Body', 'Cookies', 'Headers (4)', and 'Test Results', with 'Body' being the active tab. Under 'Body', there are three sub-tabs: 'Pretty', 'Raw', and 'Preview', with 'Pretty' being the active tab. To the right of these sub-tabs is a 'XML' dropdown with a downward arrow and a copy icon. The main content area displays the XML response:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <product>
3   <description>QualComm Snapdragon</description>
4   <id>1</id>
5   <name>iPhone7</name>
6   <price>360000</price>
7 </product>
```





APACHE CXF

VIII(C): PLAY BOOK: WEB SERVICE CLIENT

Wsdl-analyzer.com

<https://www.wsdl-analyzer.com/>





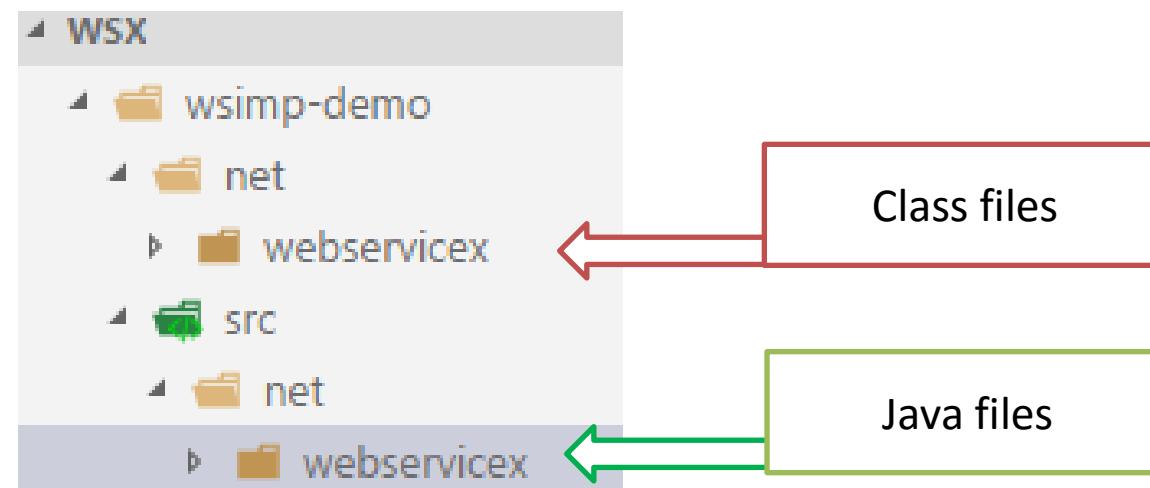
Sample wsdl services

- <http://www.thomas-bayer.com/axis2/services/BLZService?wsdl>
- <http://www.webservicex.com/CurrencyConvertor.asmx?wsdl>
- <http://www.webservicex.net/usaddressverification.asmx?WSDL>
- <http://www.webservicex.net/genericbarcode.asmx?WSDL>
- <http://www.webservicex.net/GenericNAICS.asmx?WSDL>
- <http://www.webservicex.net/GenericUNSPSC.asmx?WSDL>
- <http://www.webservicex.net/medicareSupplier.asmx?WSDL>
- <http://www.webservicex.net/FedACH.asmx?WSDL>
- <http://www.webservicex.net/FedWire.asmx?WSDL>
- <http://www.webservicex.net/WeatherForecast.asmx?WSDL>
- <http://www.webservicex.net/MortgageIndex.asmx?WSDL>
- <http://www.webservicex.net/sunsetriseservice.asmx?WSDL>

jdk: wsimport

- It is a tool to parse an existing web service description language (WSDL) file and generate required files (JAX-WS portable artifacts) for web service client to access the published web services.
- Located in **\$jdk/bin/**

```
PS C:\Program Files (x86)\Java\jdk1.7.0_71\bin> ./wsimport -d "E:\CT\JavaWebServices\codeplay\wsx\wsimp-demo" -keep -s "E:\CT\JavaWebServices\codeplay\wsx\wsimp-demo\src" http://www.webservicex.net/GenericUNSPSC.asmx?WSDL  
parsing WSDL...
```



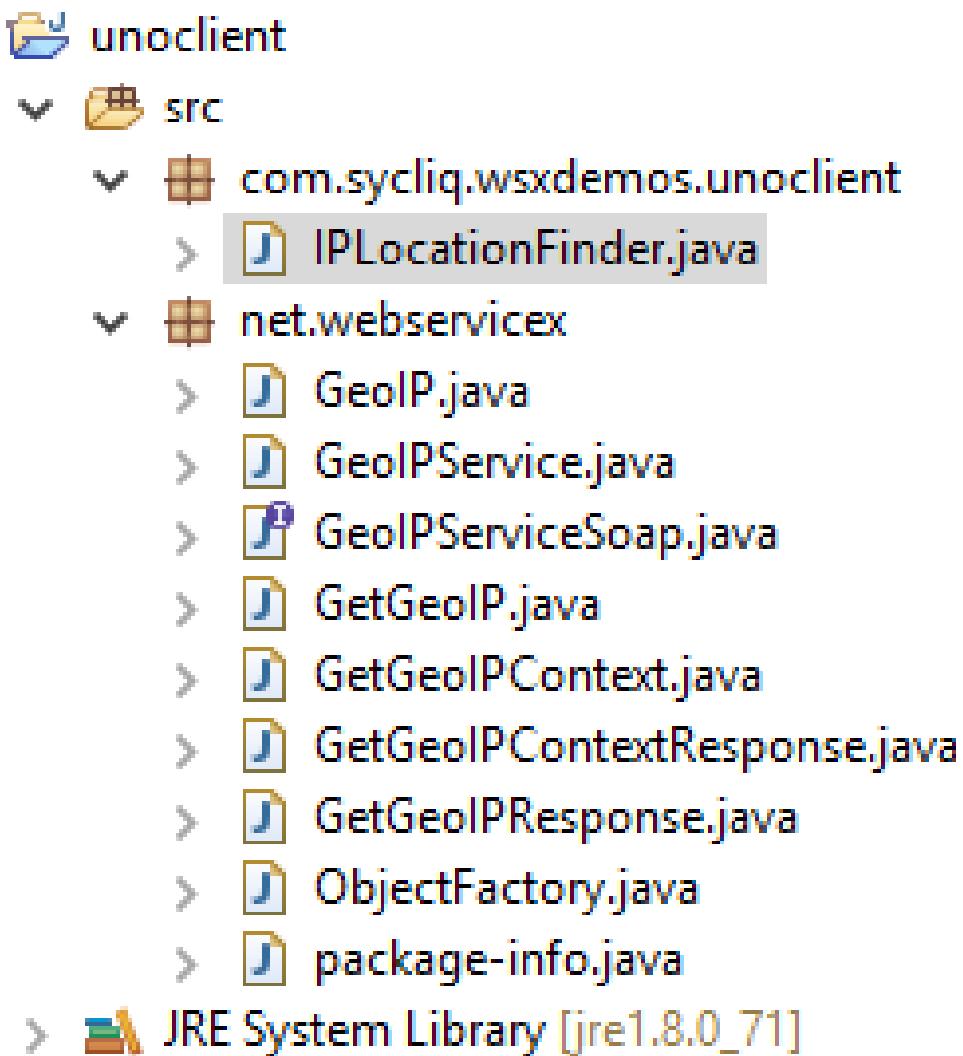
wsimport -d “directoryname” –keep –s “sourcefiledirectory” example.wsdl

```
./wsimport -d "E:\CT\JavaWebServices\codeplay\wsx\wsimp-demo" -keep -s  
"E:\CT\JavaWebServices\codeplay\wsx\wsimp-demo\src"  
http://www.webservicex.net/geoipservice.asmx?WSDL
```



Step 1:

- create java project
- Import net.webservicex
java source files post
wsimport process
- create package
com.syqliq.wsxdemox.uno
client
- IPLocationFinder.java



Step 2:

- create `IPLocationFinder` to extract the country name of the ipaddress.

```
package com.sycliq.wsxdemos.unoclient;

import net.webservicex.GeoIP;
import net.webservicex.GeoIPService;
import net.webservicex.GeoIPServiceSoap;

public class IPLocationFinder {

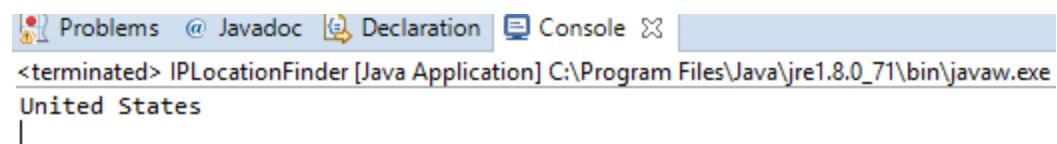
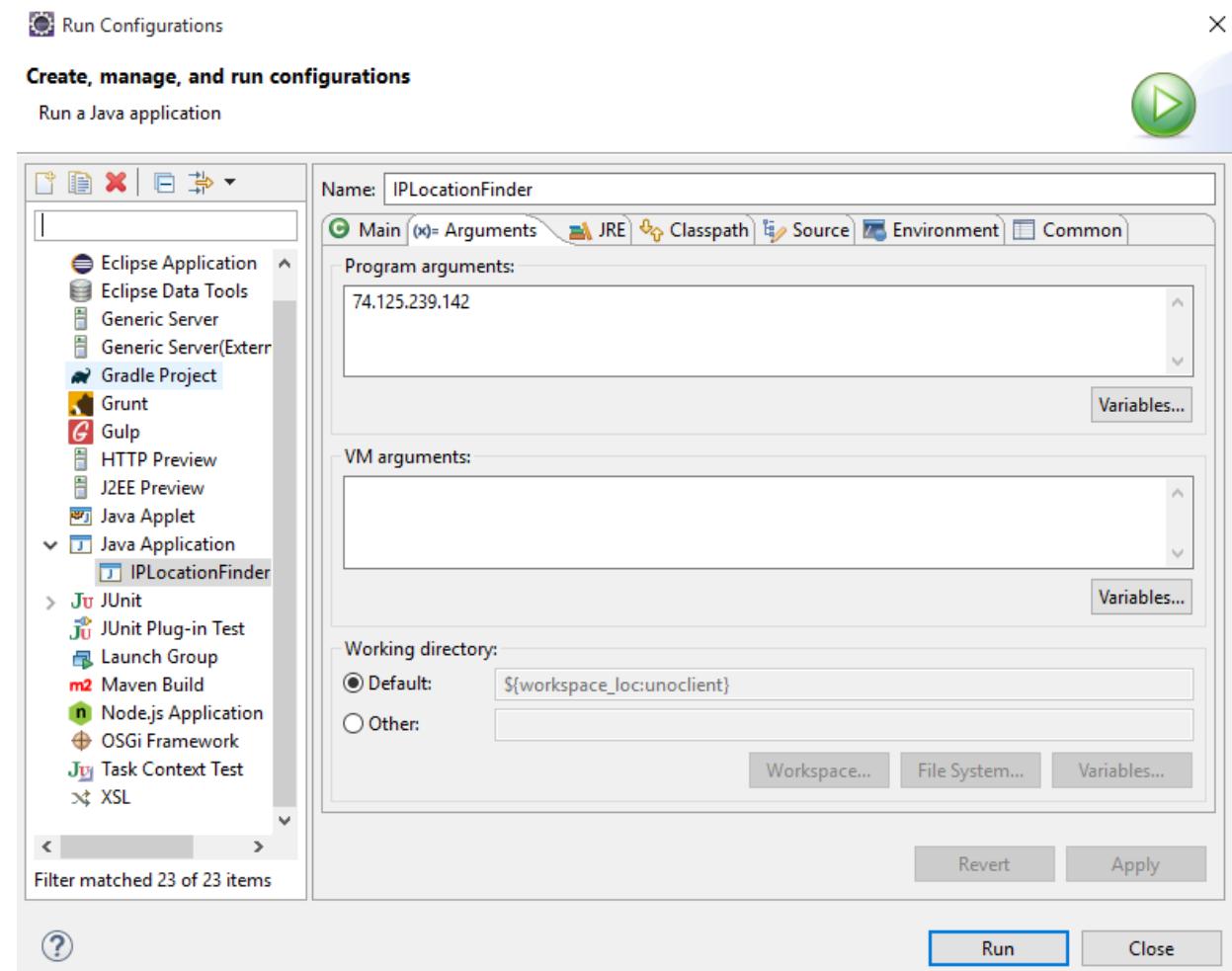
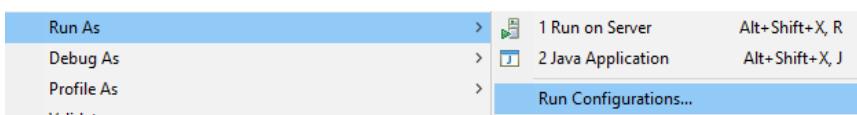
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        if(args.length !=1) {
            System.out.println("Pass on the IPAddress Please:");
        }
        else {
            String myipAddress = args[0];
            //creating an instance of the service
            GeoIPService myipService = new GeoIPService();
            // soap object from the service/port
            GeoIPServiceSoap geoIPServiceSoap = myipService.getGeoIPServiceSoap();
            // method/operations to extract the geoip
            GeoIP geoIp = geoIPServiceSoap.getGeoIP(myipAddress);
            //printing the country location
            System.out.println(geoIp.getCountryName());
        }
    }
}
```



Step 3:

- Run Configurations



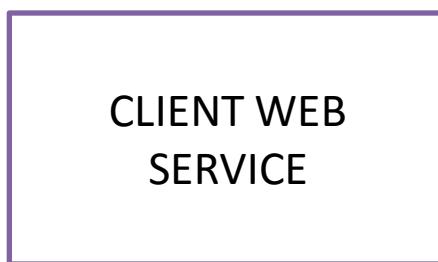
APACHE CXF

VIII(D): PLAY BOOK: CONTRACT-FIRST WEB SERVICE(JAX-WS)



CONTRACT-FIRST WEB SERVICE

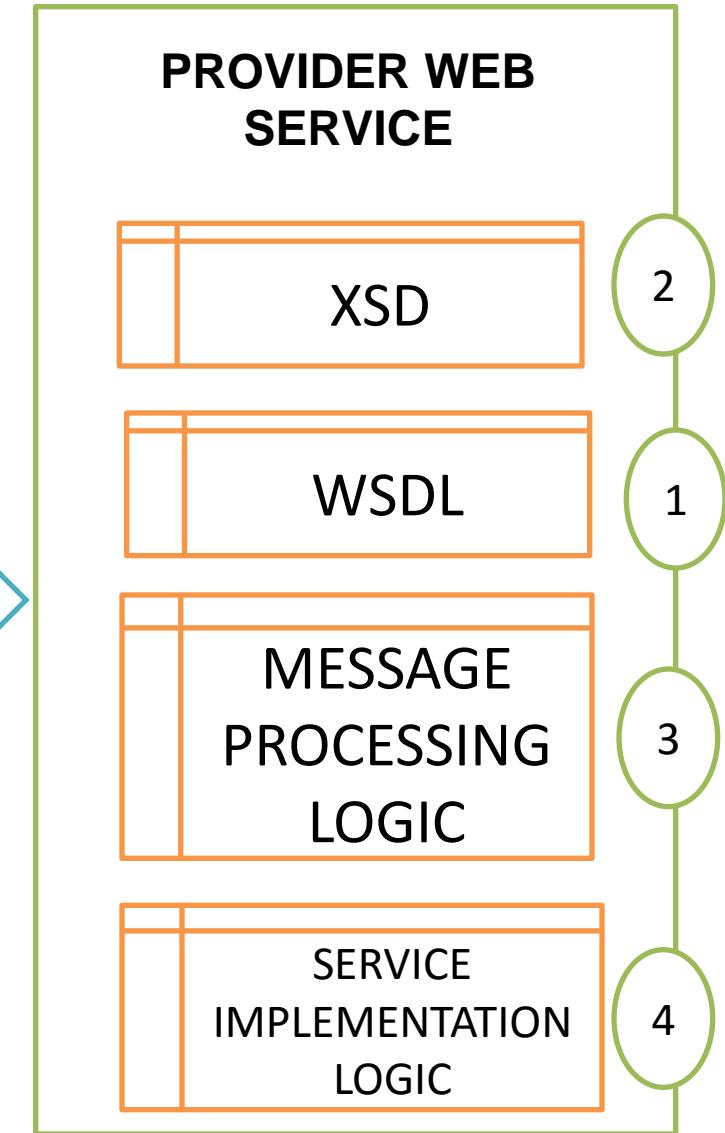
APACHE-CXF:JAX-WS



SEND MESSAGE



PROCESSED BY

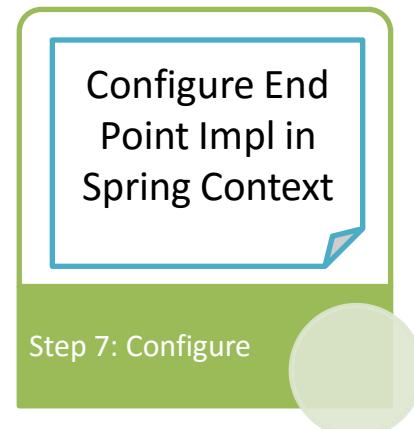
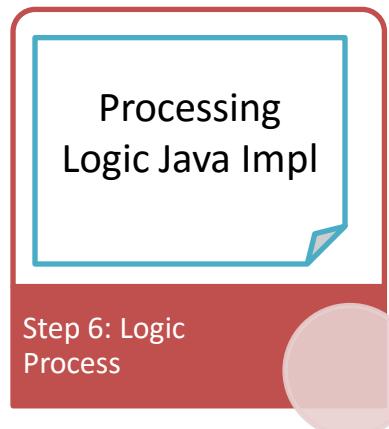
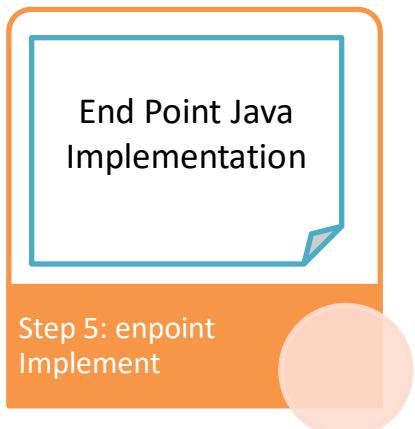
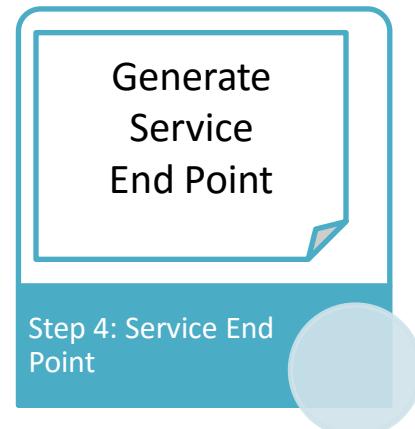
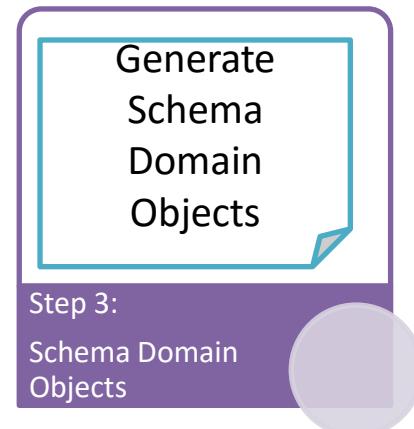
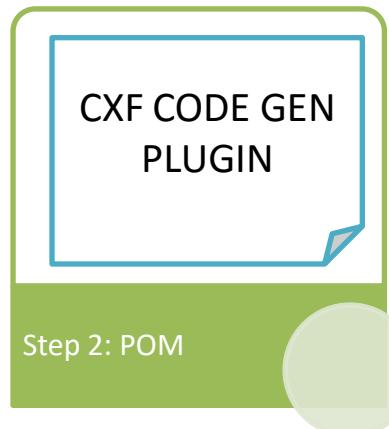


CONTRACT-FIRST WEB SERVICE?

- Inventory of services and schemas defined early in a design phase.
- Considered best practice
 - Schema types are portable
 - Supports logic-to-contract coupling
- Design considerations
 - Contract granularity
 - Breadth of business context
 - Amount of work
 - Amount of data
 - Amount of constraints
 - Re-usability
 - transparency



Web Service Implementation Steps:



Original Series



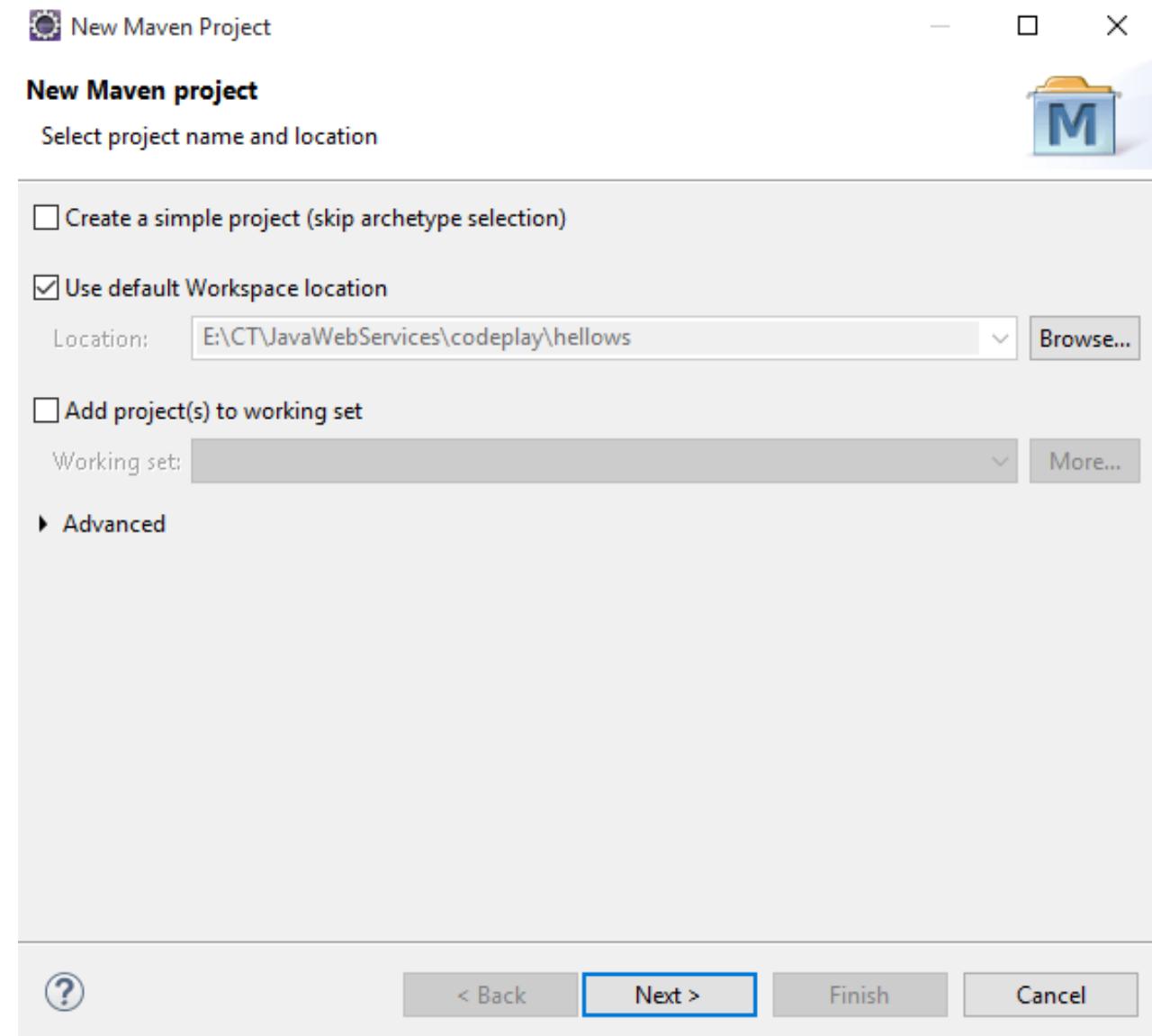
WSDL CONTRACT

- What does the contract do?
- How should the contract services be accessed?
- Where can the contract services be accessed from ?



STEP 1

- **CREATE A NEW MAVEN PROJECT USING DEFAULT WORKSPACE LOCATION.**

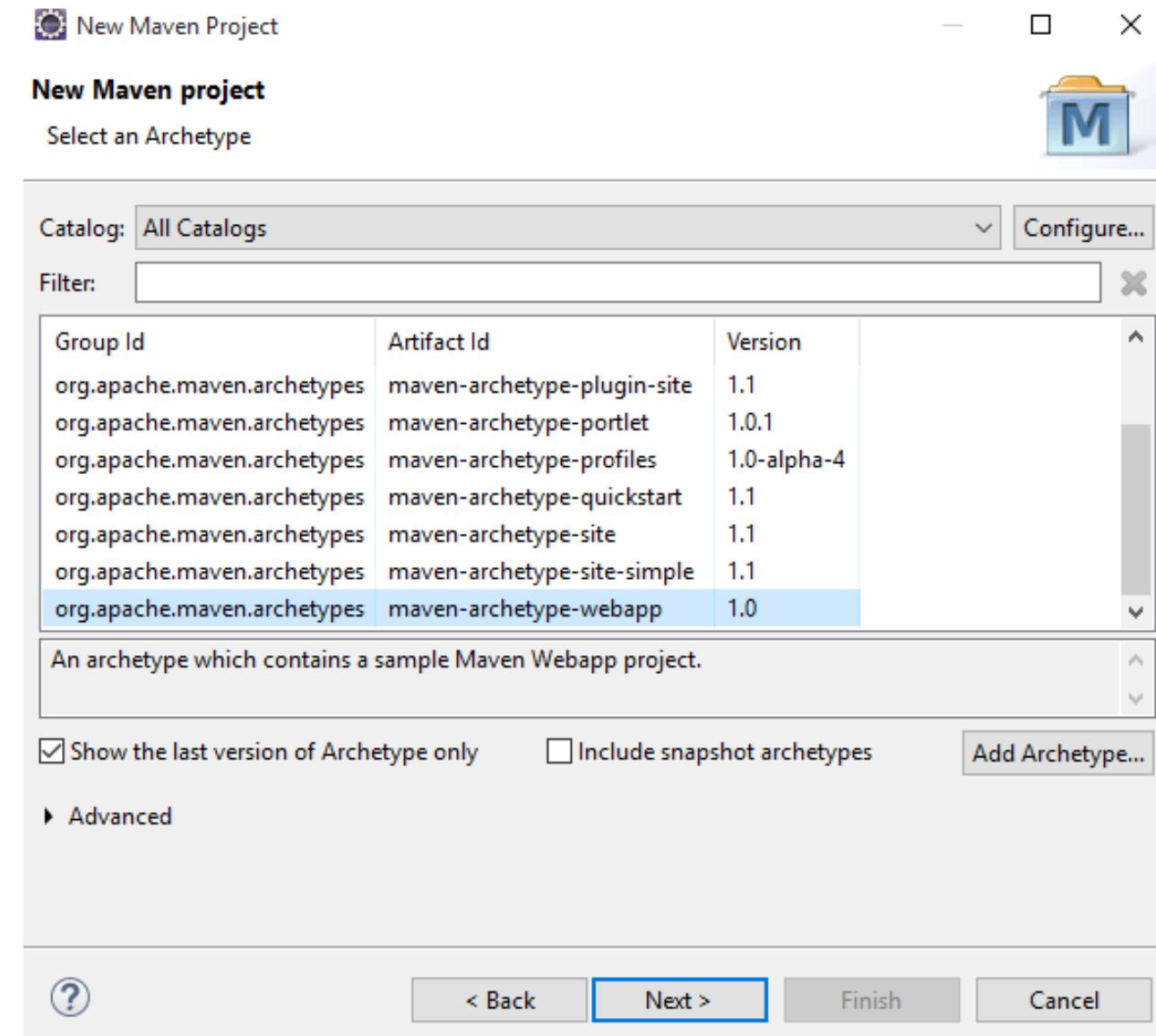


STEP 2

- SELECT
ORG.APACHE.MAVEN.
ARCHETYPES =>
**MAVEN.ARCHETYPE-
WEBAPP**

MAVEN ARCHETYPE

An archetype is a templating tool that standardizes the folder structure and artifacts of a project build.



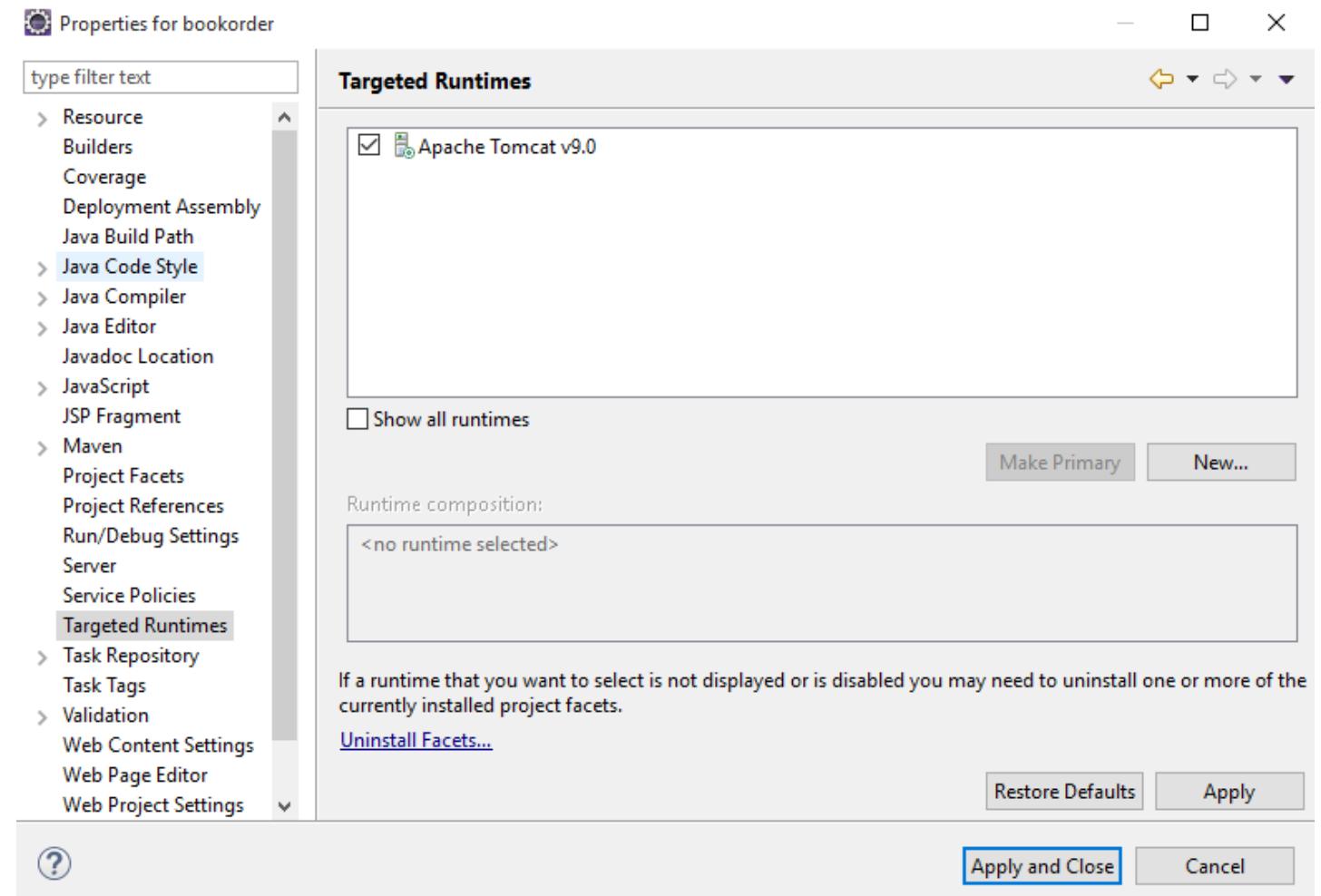


Step 3:

- Create an Maven project with
 - groupId: com.trpi.cfdemoone
 - artifactId: bookorder

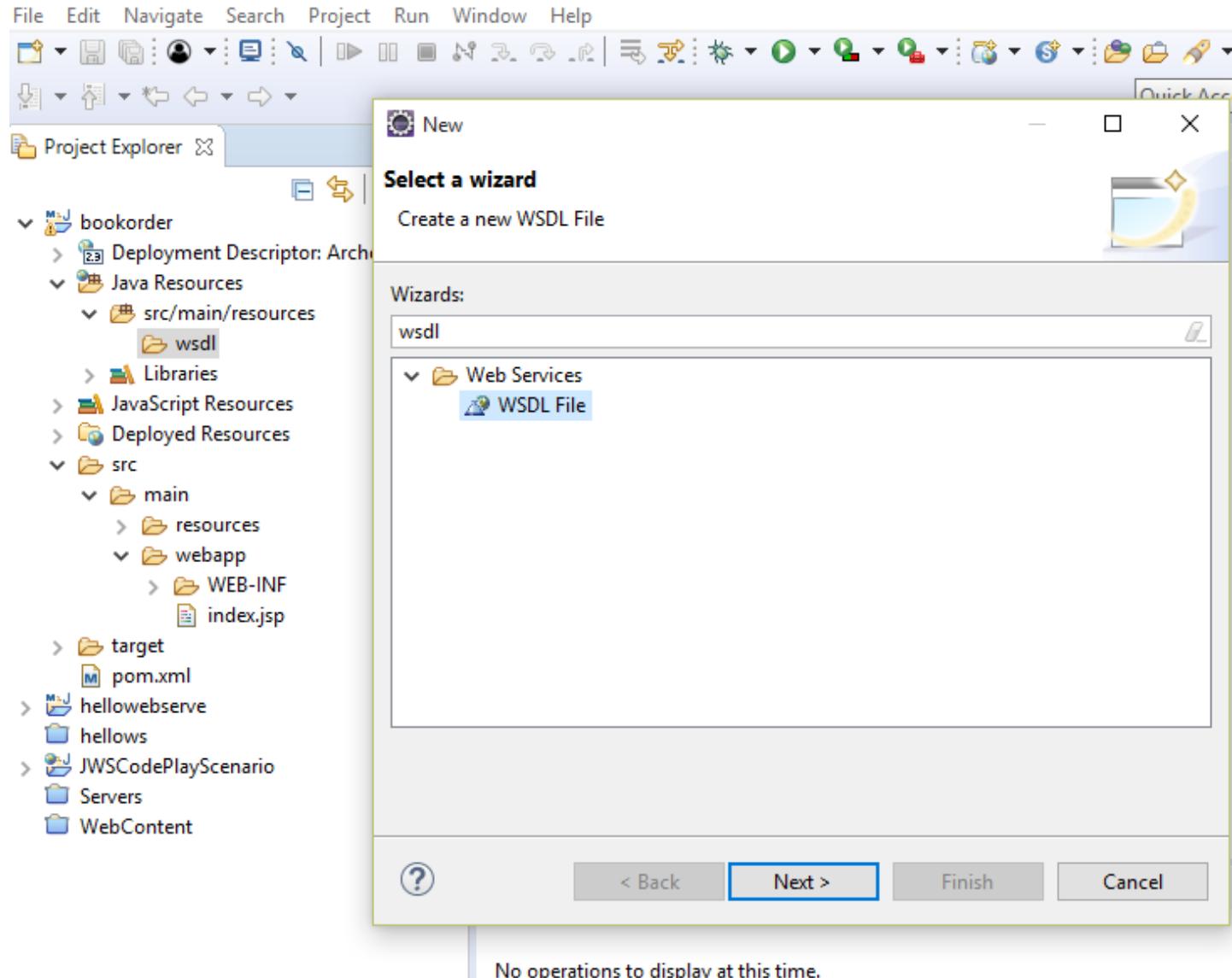
Step 4:

- Set the target run-time to your current jdk version



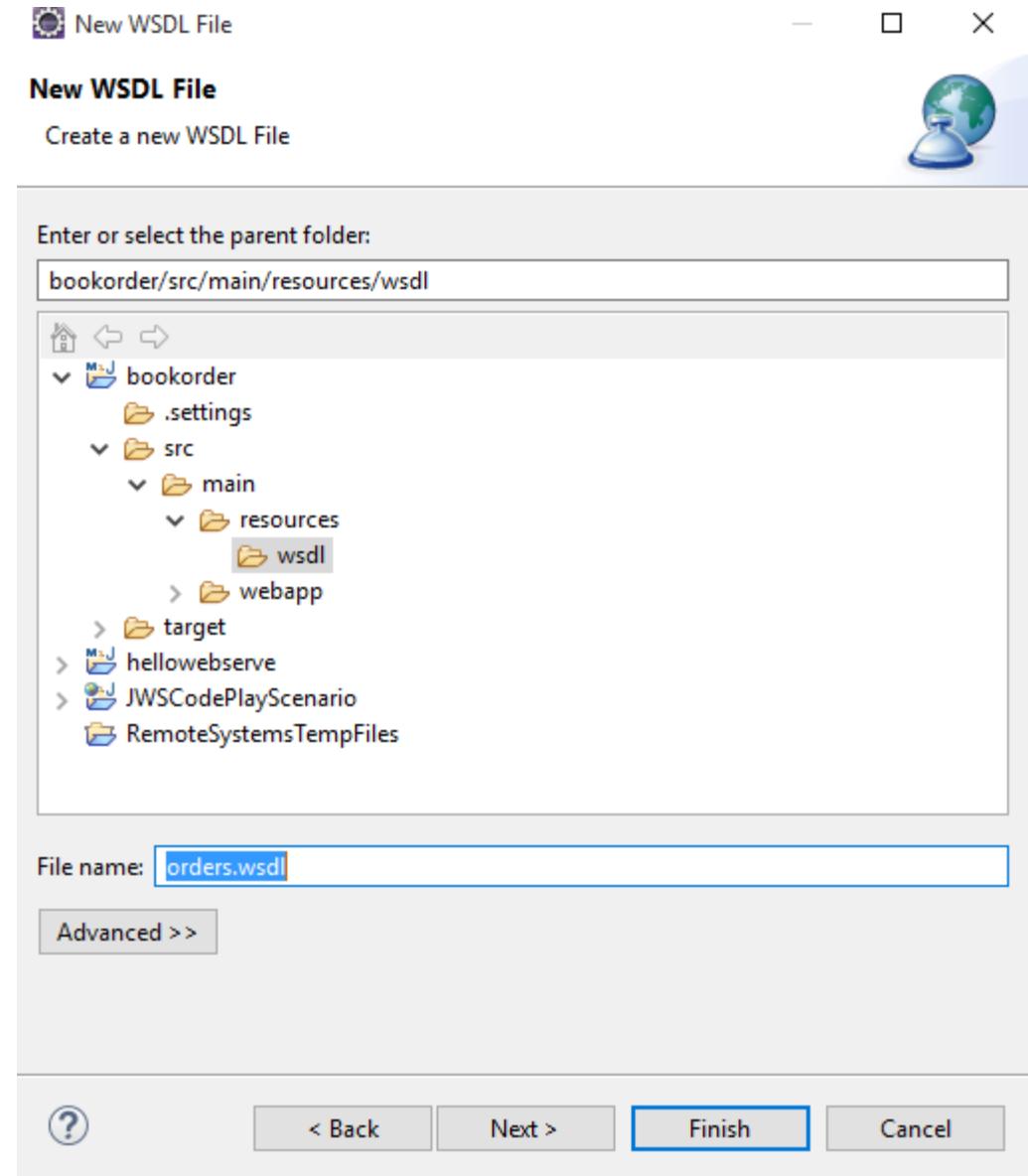
Step 5

- create a wsdl folder and wsdl file @
src/main/resources/wsdl/
orders.wsdl



Step 6

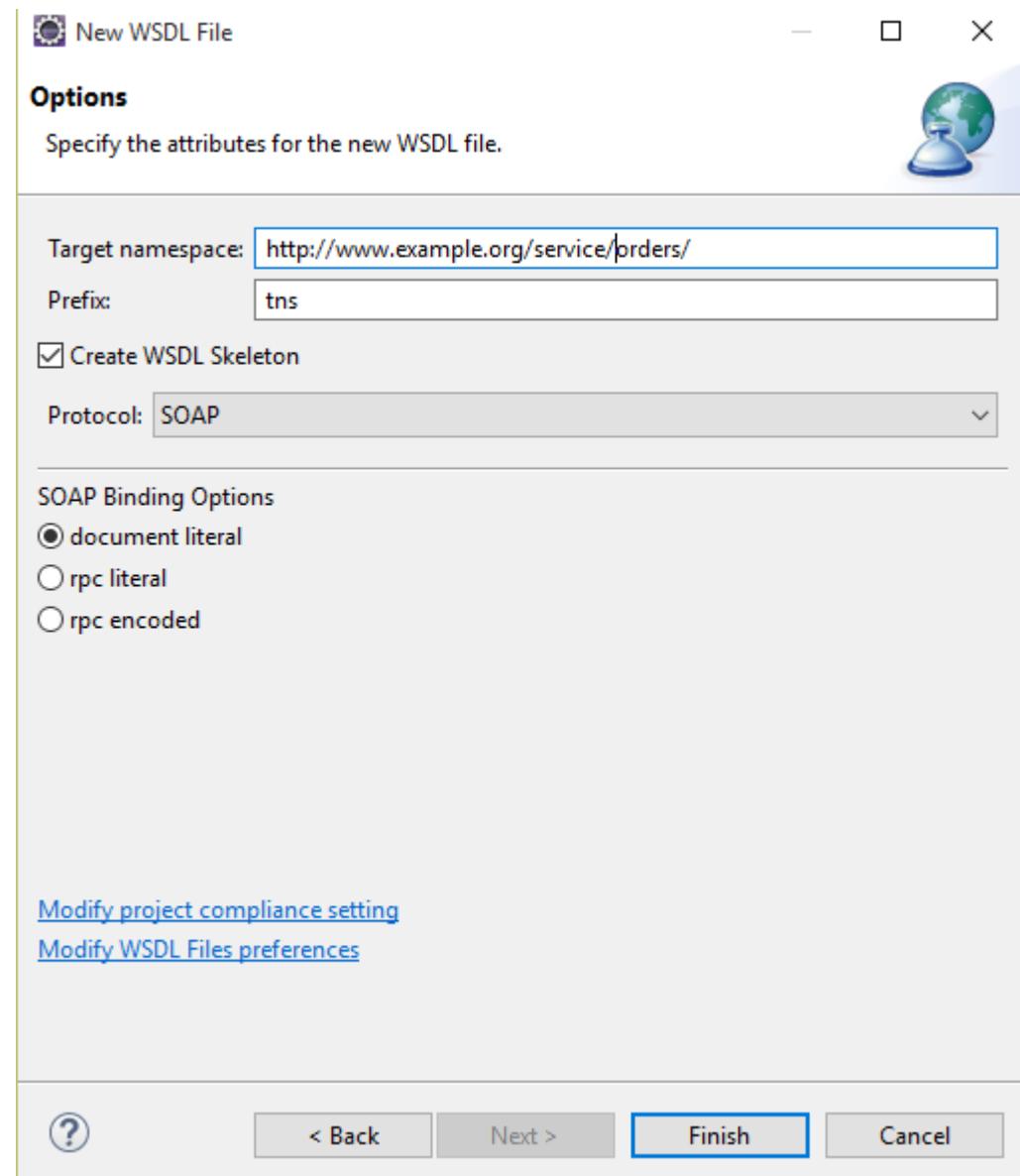
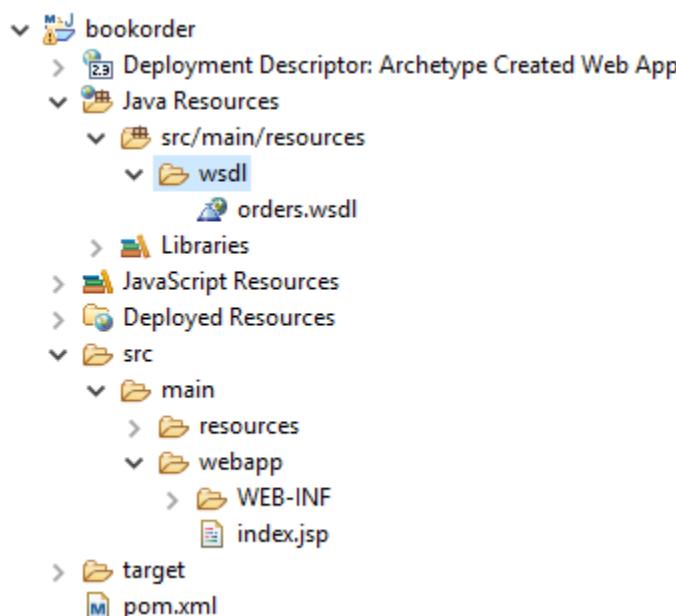
- create orders.wsdl file





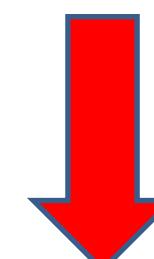
Step 7:

- define the wsdl options outlined
- Qualified target namespace
`http://www.example.org/services/orders/`



```
orders.wsdl X
15     <xsd:element name="out" type="xsd:string"/>
16   </xsd:sequence>
17 </xsd:complexType>
18 </xsd:element>
19 </xsd:schema>
20 </wsdl:types>
21<wsdl:message name="NewOperationRequest">
22   <wsdl:part element="tns:NewOperation" name="parameters"/>
23 </wsdl:message>
24<wsdl:message name="NewOperationResponse">
25   <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
26 </wsdl:message>
27<wsdl:portType name="orders">
28   <wsdl:operation name="NewOperation">
29     <wsdl:input message="tns:NewOperationRequest"/>
30     <wsdl:output message="tns:NewOperationResponse"/>
31   </wsdl:operation>
32 </wsdl:portType>
33<wsdl:binding name="ordersSOAP" type="tns:orders">
34   <soap:binding style="document" transport="http://schemas.xmlsoap.o
35 <wsdl:operation name="NewOperation">
36   <soap:operation soapAction="http://www.example.org/service/order
37 <wsdl:input>
38   <soap:body use="literal"/>
39 </wsdl:input>
40 <wsdl:output>
41   <soap:body use="literal"/>
42 </wsdl:output>
43 </wsdl:operation>
44 </wsdl:binding>
```

} Comparable to interface in JAVA

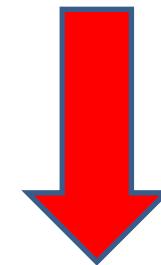


```
<wsdl:portType name="orders">
  <wsdl:operation name="ProcessOrderPlacement">
    <wsdl:input message="tns:ProcessOrderPlacementRequest"/>
    <wsdl:output message="tns:ProcessOrderPlacementResponse"/>
  </wsdl:operation>
</wsdl:portType>
```



```
11      </xsd:element>
12      <xsd:element name="NewOperationResponse">
13          <xsd:complexType>
14              <xsd:sequence>
15                  <xsd:element name="out" type="xsd:string"/>
16              </xsd:sequence>
17          </xsd:complexType>
18      </xsd:element>
19  </xsd:schema>
20</wsdl:types>
21<wsdl:message name="NewOperationRequest">
22    <wsdl:part element="tns:NewOperation" name="parameters"/>
23</wsdl:message>
24<wsdl:message name="NewOperationResponse">
25    <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
26</wsdl:message>
27<wsdl:portType name="orders">
28    <wsdl:operation name="ProcessOrderPlacement">
29        <wsdl:input message="tns:ProcessOrderPlacementRequest"/>
30        <wsdl:output message="tns:ProcessOrderPlacementResponse"/>
31    </wsdl:operation>
32</wsdl:portType>
33<wsdl:binding name="ordersSOAP" type="tns:orders">
34    <soap:binding style="document" transport="http://schemas.xmlsoap.o
35    <wsdl:operation name="NewOperation">
36        <soap:operation soapAction="http://www.example.org/service/order
37    <wsdl:input>
38        <soap:body use="literal"/>
39    </wsdl:input>
40    <wsdl:output>
```

Correspond to the operations in the port type

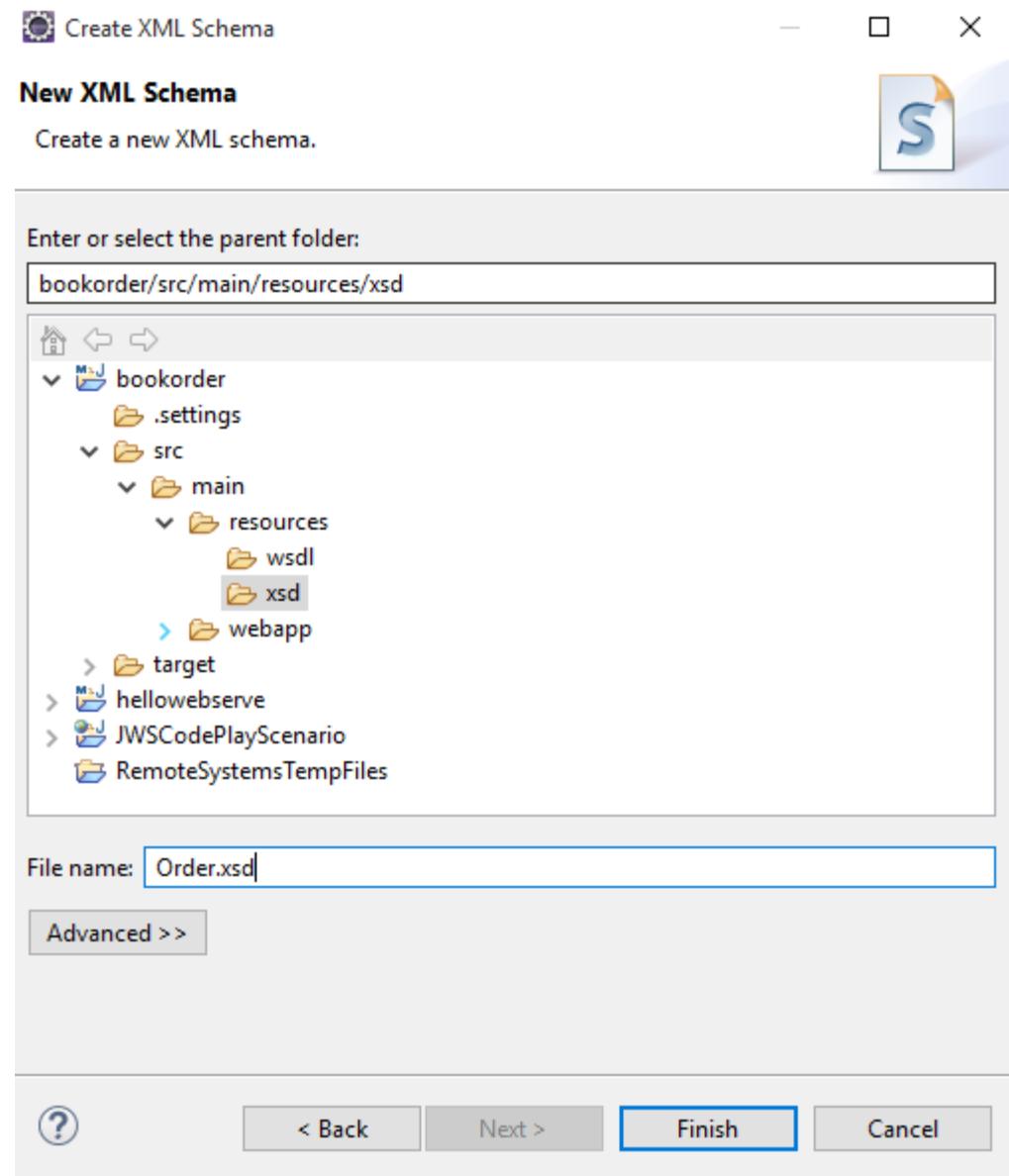


```
<wsdl:message name="ProcessOrderPlacementRequest">
    <wsdl:part element="tns:NewOperation" name="parameters"/>
</wsdl:message>
<wsdl:message name="ProcessOrderPlacementResponse">
    <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
</wsdl:message>
```



Step 8

- define the schema structure for order service
=> order.xsd



Step 9

- Subsequently replace the schema elements in orders.wsdl file by importing the order.xsd file

```
<xsd:schema targetNamespace="http://www.example.org/service/orders/">
    <xsd:import namespace="http://www.example.org/schema/Order"
        schemaLocation="..../xsd/Order.xsd"></xsd:import>
</xsd:schema>
```



Step 10

- configuring the wsdl file

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:order="http://www.example.org/schema/Order" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.org/service/orders/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="orders" targetNamespace="http://www.example.org/service/orders/">
    <wsdl:types>
        <xsd:schema targetNamespace="http://www.example.org/service/orders/">
            <xsd:import namespace="http://www.example.org/schema/Order"
                schemaLocation="../xsd/Order.xsd"></xsd:import>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="ProcessOrderPlacementRequest">
        <wsdl:part element="order:orderInquiry" name="orderInquiry" />
    </wsdl:message>
    <wsdl:message name="ProcessOrderPlacementResponse">
        <wsdl:part element="order:orderInquiryResponse" name="orderInquiryResponse" />
    </wsdl:message>
    <wsdl:portType name="orders">
        <wsdl:operation name="ProcessOrderPlacement">
            <wsdl:input message="tns:ProcessOrderPlacementRequest" />
            <wsdl:output message="tns:ProcessOrderPlacementResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="ordersSOAP" type="tns:orders">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="ProcessOrderPlacement">
            <soap:operation soapAction="http://www.example.org/service/orders/ProcessOrderPlacement" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="orders">
        <wsdl:port binding="tns:ordersSOAP" name="ordersSOAP">
            <soap:address location="http://localhost:8080/bookorder/services/orders" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```



Step 11

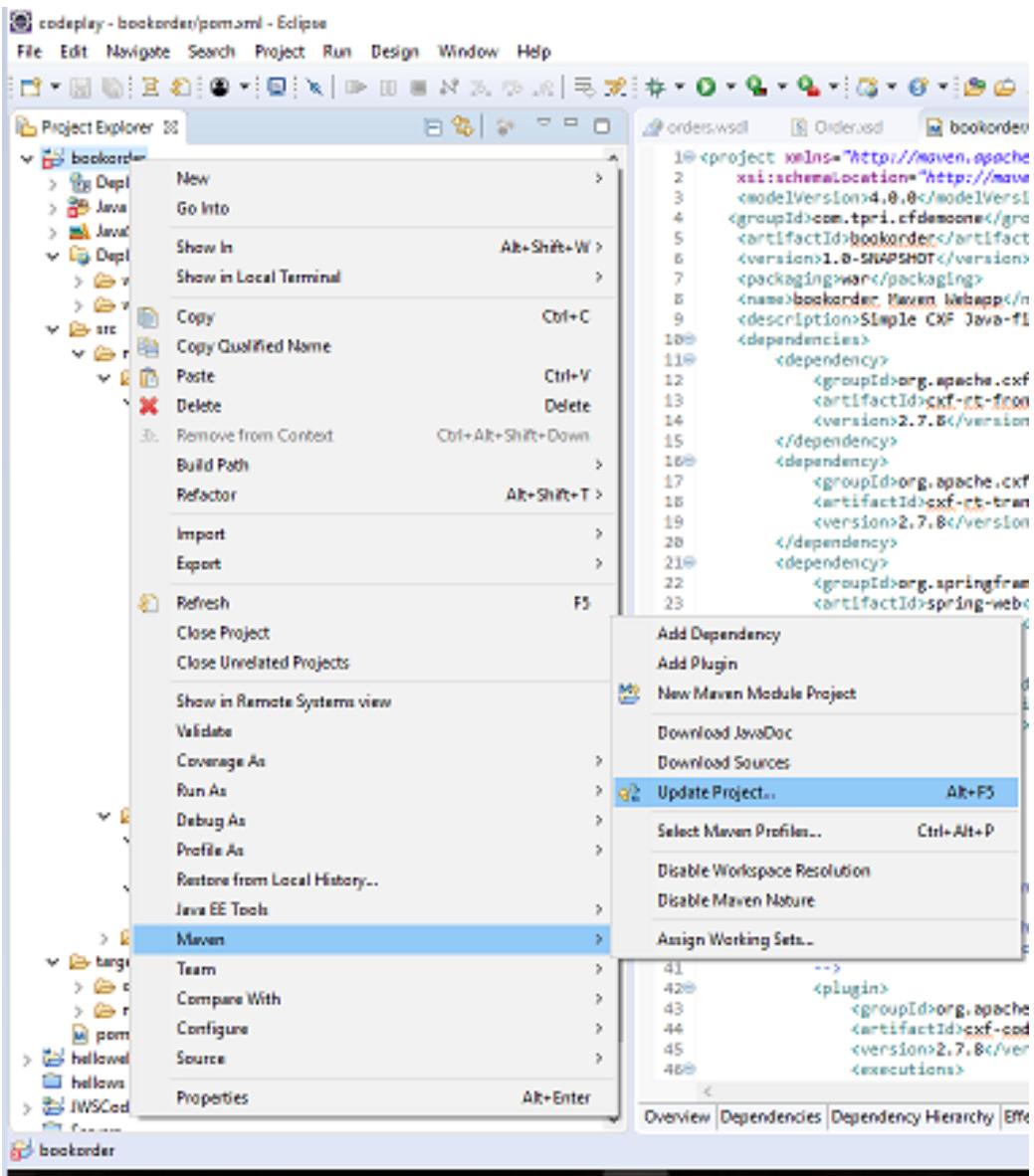
- Update the POM.xml with Maven Plugins and dependencies.

```
1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3  <modelVersion>4.0.0</modelVersion>
4  <groupId>com.tpri.cfdeomoone</groupId>
5  <artifactId>bookorder</artifactId>
6  <version>1.0-SNAPSHOT</version>
7  <packaging>war</packaging>
8  <name>bookorder Maven Webapp</name>
9  <description>Simple CXF Java-first SOAP project using Spring configuration</description>
10 <dependencies>
11   <dependency>
12     <groupId>org.apache.cxf</groupId>
13     <artifactId>cxf-rt-frontend-jaxws</artifactId>
14     <version>2.7.8</version>
15   </dependency>
16   <dependency>
17     <groupId>org.apache.cxf</groupId>
18     <artifactId>cxf-rt-transports-http</artifactId>
19     <version>2.7.8</version>
20   </dependency>
21   <dependency>
22     <groupId>org.springframework</groupId>
23     <artifactId>spring-web</artifactId>
24     <version>3.0.7.RELEASE</version>
25   </dependency>
26   <dependency>
27     <groupId>junit</groupId>
28     <artifactId>junit</artifactId>
29     <version>4.11</version>
30     <scope>test</scope>
31   </dependency>
32 </dependencies>
33
34 <build>
35   <plugins>
36     <!--
37       Apache CXF Code Generation Plugin. This plug-in will generate code for the
38       wsdl specified. We want to execute WSDL data binding as part of Maven's
39       generate sources phase. After you add this, you will need to update
40       the project Maven configuration, then run the generate-sources goal.
41     -->
42     <plugin>
43       <groupId>org.apache.cxf</groupId>
44       <artifactId>cxf-codegen-plugin</artifactId>
45       <version>2.7.8</version>
46       <executions>
```



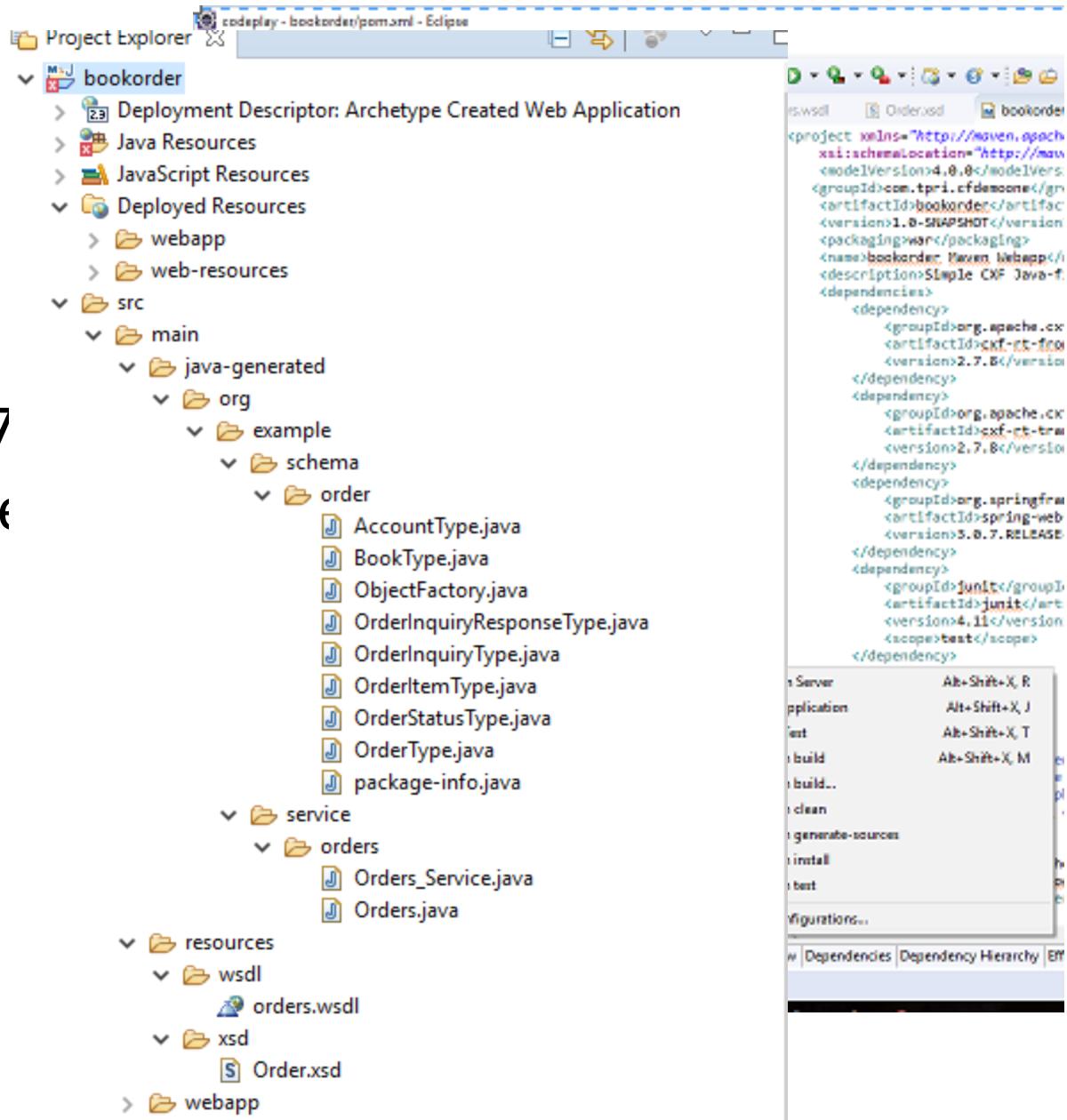
Step 12

- Update the Maven Project (Alt+F5)
- Run as => Maven-Clean



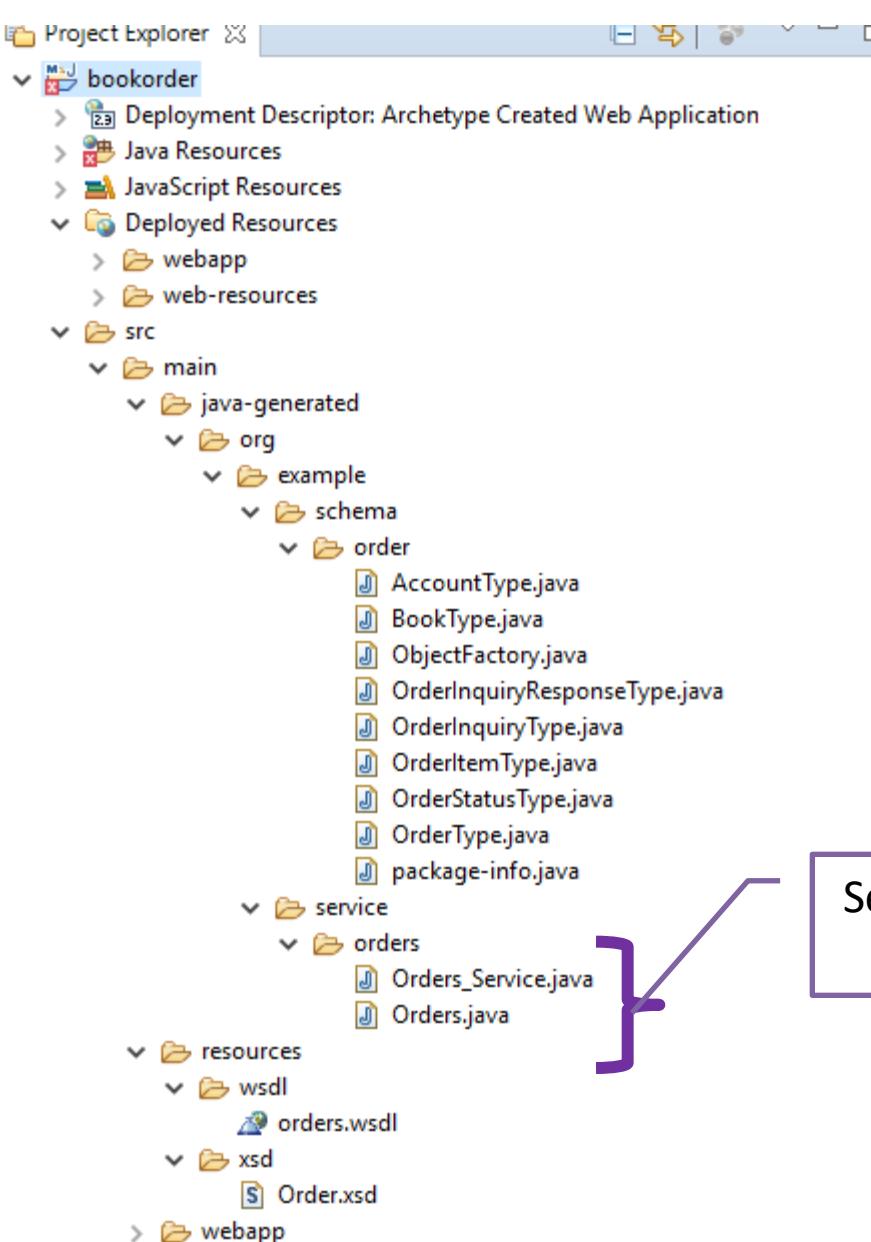
Step 13

- Project => Run As => 6. Maven Clean
- Project => Run As => 7. Maven generate-sources

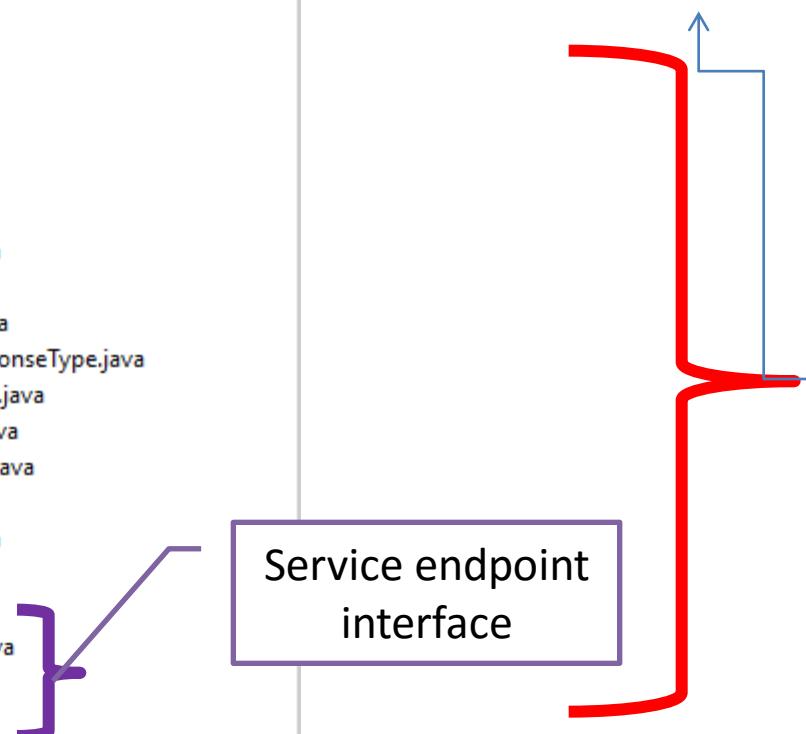




- wsdl2java generates java schema files and service files as shown in the figure



Created post
Maven generate-
sources



JAX-WS Endpoint Bean Configuration

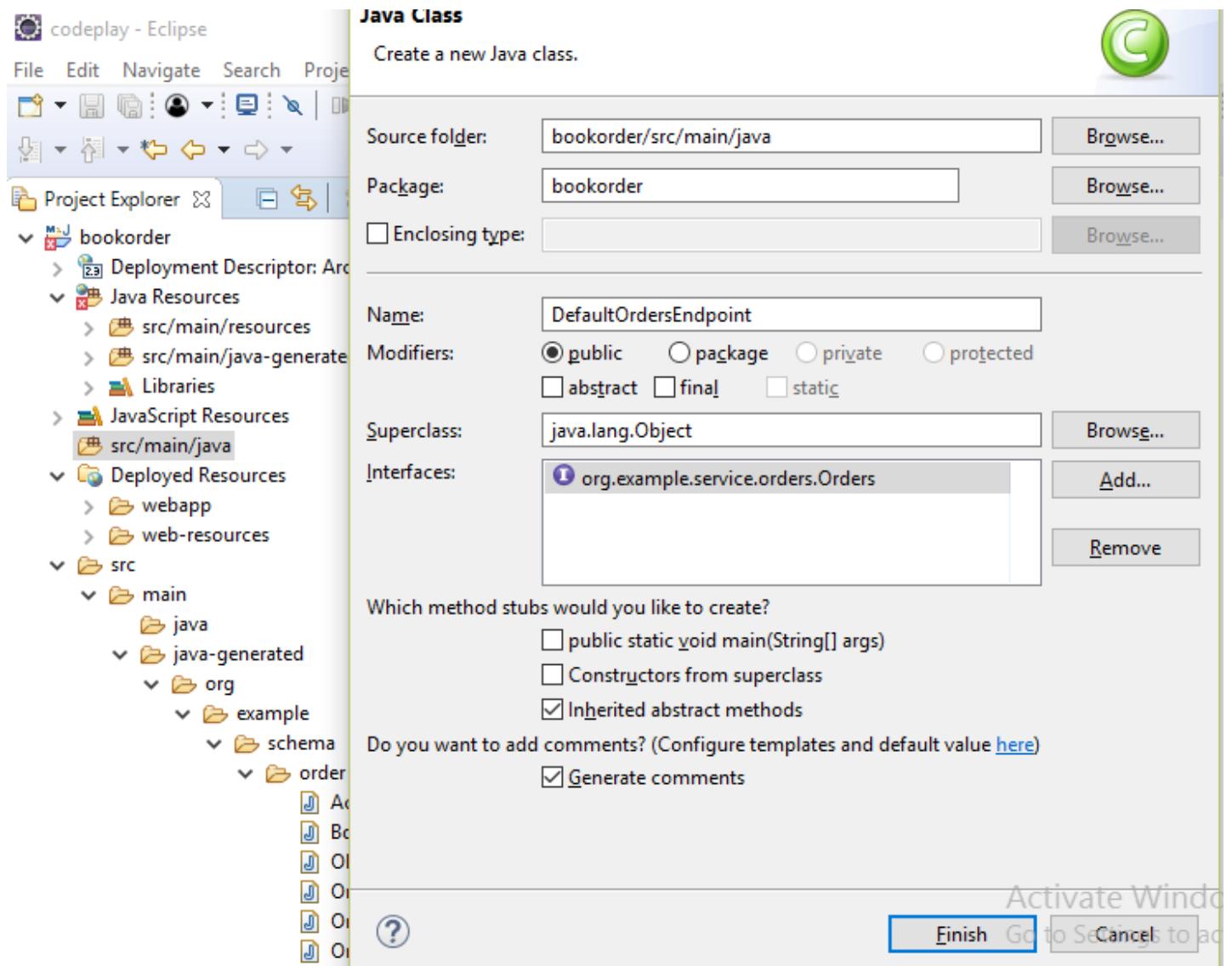
```
<jaxws:endpoint  
address=""  
id=""  
implementor=""  
bindingUri=""  
bus=""  
endpointName=""  
implementorClass=""  
publish=""  
publishedEndpointUrl=""  
serviceName=""  
transportId=""  
wsdlLocation=""  
/>
```

```
<!--child properties of jaxws:endpoint-->  
<jaxws:endpoint>  
  <jaxws:dataBinding/>  
  <jaxws:features/>  
  <jaxws:handlers/>  
  <jaxws:inInterceptors/>  
  <jaxws:inFaultInterceptors/>  
  <jaxws:outInterceptors/>  
  <jaxws:outFaultInterceptors/>  
  <jaxws:properties/>  
</jaxws:endpoint>
```



Step 14

- Create **DefaultOrderEndpoint** by implementing the orders interface



Step 15

- The DefaultEndPoint Service

```
1 package bookorder;
2
3 import javax.jws.WebService;
4
5 import org.example.schema.order.AccountType;
6 import org.example.schema.order.OrderInquiryResponseType;
7 import org.example.schema.order.OrderInquiryType;
8 import org.example.service.orders.Orders;
9 import org.example.schema.order.ObjectFactory;
10
11 @WebService(portName = "ordersSOAP",
12 serviceName = "orders",
13 endpointInterface = "www.example.org/service/orders/")
14 public class DefaultOrdersEndpoint implements Orders {
15
16     @Override
17     public OrderInquiryResponseType processOrderPlacement(OrderInquiryType
18     orderInquiry) {
19         ObjectFactory factory = new ObjectFactory();
20         OrderInquiryResponseType response = factory.createOrderInquiryResponseType();
21         AccountType account = factory.createAccountType();
22         account.setAccountId(1);
23         response.setAccount(account);
24
25         return response;
26     }
27 }
```



Step 16:

- configure the beans.xml with jaxws:endpoint and spring context :component-scan
- Maven-clean
- Maven-install

The screenshot shows a Java project structure in an IDE. The project contains several Java files (BookType.java, ObjectFactory.java, OrderInquiryRe..., OrderInquiryTy..., OrderItemType.java, OrderStatusTyp..., OrderType.java, package-info.java) and a package-info.java file. It also includes service, resources, wsdl, xsd, webapp, and target folders. The beans.xml file is located in the WEB-INF folder of the webapp. The code editor on the right displays the beans.xml configuration, which includes XML declarations for namespaces, imports, component scanning, and a JAXWS endpoint definition.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2<beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:context="http://www.springframework.org/schema/context"
5   xmlns:jaxws="http://cxf.apache.org/jaxws"
6   xsi:schemaLocation="
7       http://www.springframework.org/schema/context
8       http://www.springframework.org/schema/beans
9       http://www.springframework.org/schema/beans/spring-beans.xsd
10      http://cxf.apache.org/jaxws
11      http://cxf.apache.org/schemas/jaxws.xsd">
12
13<jaxws:endpoint id="orders"
14   implementor="bookorder.DefaultOrdersEndpoint"
15   address="/services/orders">
16</jaxws:endpoint>
17</beans>
```



Step 17

- run the tomcat web server





SYED AWASE

IX: REST: REPRESENTATIONAL STATE TRANSFER

REST

An architectural style that can help you build **client-friendly** distributed systems that are **simple to understand** and **simple to scale**

- Roy Fielding, 2000, Doctoral Thesis.



RPC

- Contract is a service and its operations
- Action semantics are specified out of band
- Error semantics are specified out of band
- Limited caching support
- Client and server share ownership of the URL namespace.
- Input and outputs are tied to underlying runtime types
- Can tunnel through multiple protocols

REST

- Contract is the uniform interface
- Actions semantics are specified by the uniform interface and state transitions are specified by hypermedia controls embedded in representations
- Error semantics are specified by the uniform interface
- Caching supported by all intermediaries which understand the uniform interface
- Server owns the URL namespace
- Inputs and outputs are ties to the media type specification
- Tied to the uniform interface of the supporting protocol



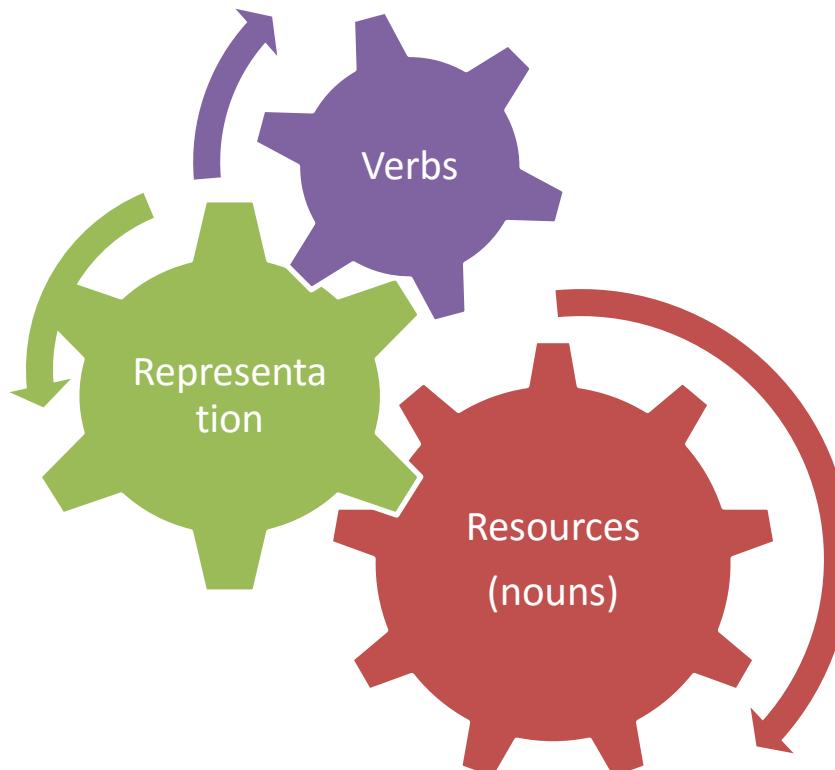
REST: Representational State Transfer

- An architectural style, which uses existing HTTP actions and methods and does not create any new standards.
- REST supports different formats and supports messages which are smaller in size and consume lesser bandwidth.
- REST is better in terms of performance with better caching support.
- No third party tool is required to access REST web services.
- There is less coupling between REST Clients and Servers, feature extensions and changes can be made easily.
- It is adopted when there is a need for highly secure and complex API, which supports different protocols.



REST

- Ideal choice when there is a need for developing lightweight APIs with high performance and support for CRUD operations.



- Resources:** Fundamental elements of the web platform. Every resource has a unique identifier on the web platform which is known as the universal resource identifier (URI)
- Representation:** It is determining a way to showcase the resources to the clients. REST supports all formats without any restrictions.
- Verb:** A verb is an HTTP action like POST,PUT,GET,DELETE, OPTIONS, ETC. HTTP Verbs only tell you which action needs to be performed on the host. There are many actions that a client can trigger on the host.



REST

RESOURCES

REPRESENTATIONS

OPERATIONS

HYPertext

STATELESSNESS



RESOURCES

- Every “thing”, every “resource” get a unique identifier.
- Resources are **nouns**
- URL is the unified concept of a unique identifier on the web
 - It identifies a resource by location, name or both
 - A URL is a subset of a URI. It specifies where a resource lives and the mechanism to retrieve it.
- REST says to **identify everything that is worth being identified by your clients.**
 - /tpri/aspire/university/program
 - /tpri/syqliq/crop
 - /tpri/syqliq/commuting/car
- **Collection of things merit identification**
- **Resources can be static or change over time.**



REPRESENTATIONS

- RESTful systems empower clients to ask for data in a form that they understand.

▼ General

Request URL: <https://raw.githubusercontent.com/awasekhirni/jsondata/master/apple-gsmarena.json>

Request Method: GET

Status Code: 200 OK

Remote Address: 151.101.0.133:443

Referrer Policy: no-referrer-when-downgrade

▼ Request Headers view source

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cache-Control: max-age=0
Connection: keep-alive
Host: raw.githubusercontent.com
If-None-Match: "6a70cc714f1ac654ea935fcf689544d18b0cf99d"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
o) Chrome/61.0.3163.100 Safari/537.36
```



REPRESENTATIONS

- The type/form/representation of data requested is specified as a MIME type (e.g. text/html). There are over 1000 standard MIME types.
- Developers frequently define different URLs for different representations of the same resources
 - <http://sycliq.com/crop/2013/yield.xml>
 - <http://sycliq.com/crop/2013/yield.json>
- REST encourages to create one identifier that can be rendered in different forms
- A RICH REST ECOSYSTEM can enable us to create a negotiable request and response that is flexible to favour the client's requirement at runtime.



OPERATIONS

- REST defines different operations, which are mapped on the HTTP action verbs
 - GET: retrieve a representation of the resource
 - PUT: create a resource at a known URI or update an existing resource by replacing it
 - POST:create a resource when you don't know URL, partial update a resource, invoke arbitrary processing.
 - DELETE: delete a resource
- Client request often is mapped to a specific HTTP action verb.
- GET <http://syqliq.com/crop/yield/2013>
- All resources support one or more of these operations.



OPERATIONS:RULES

- GET:
 - Safe and idempotent- it modifies nothing and there is no adverse when called multiple times
- DELETE:
 - Not safe, but idempotent- it modifies resources, but there is no additional effect when called multiple times.
 - Subsequent calls are ignored by the server
- PUT:
 - Not safe, but idempotent- it modifies resources, but there is no additional effect when called multiple times with the same parameters.
 - Subsequent calls to the URI with the same body have no effect on the system.
- POST:
 - Not safe:Not idempotent- it modifies resources and multiple calls will cause additional effect on the system.
 - Service returns a resource to track it.
 - Server can send an error code.

Read	Write
GET	PUT,POST,DELETE



HYPertext

- In REST, application state is transferred and discovered within hypertext responses.
- A REST client needs less knowledge about how to interact with any particular service compared to clients that interact with operation-centric services
- A client transitions through application states by selecting appropriate URI and the parameters defined with it in the query string.



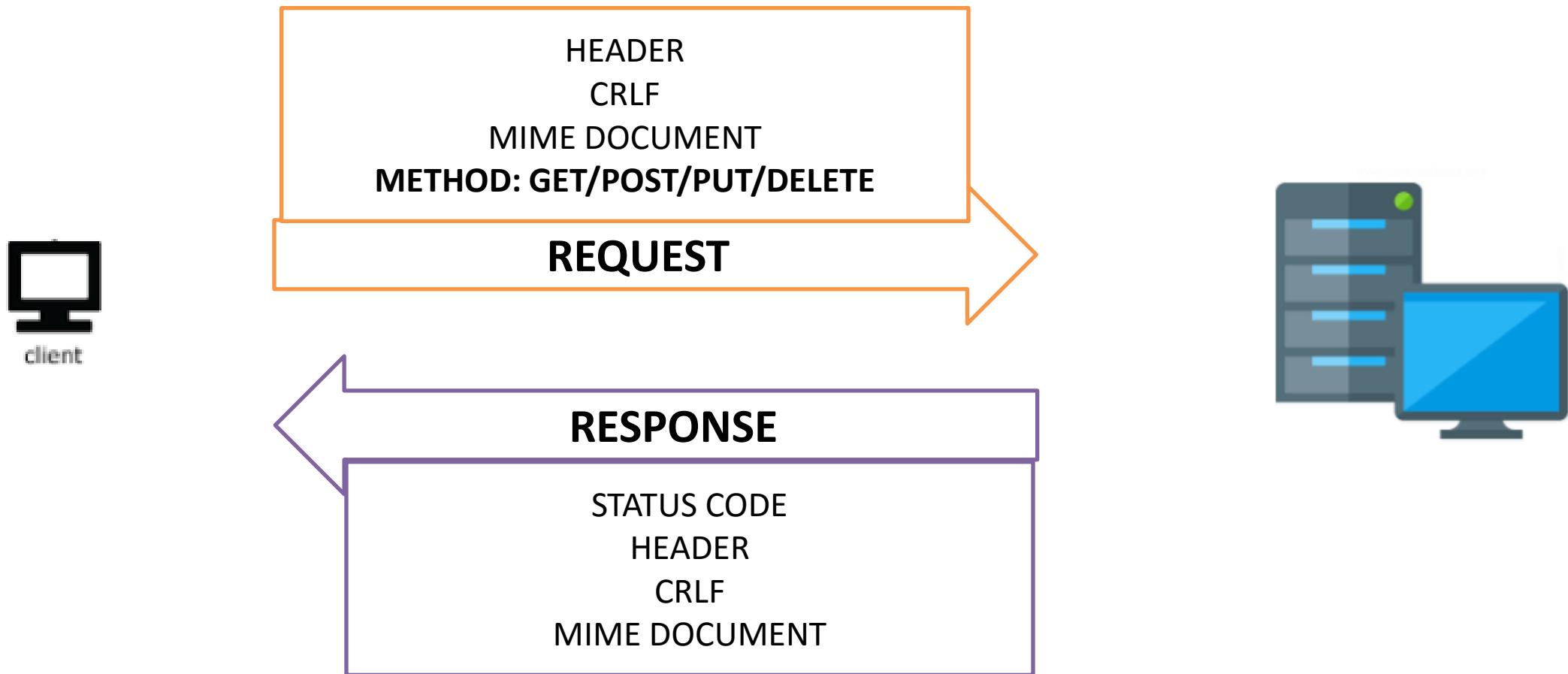
STATELESSNESS

- REST mandates that state either be turned into resource state or kept on the client. The client is responsible for application state. The server is responsible for resource state.
- The server should not retain some sort of communication state for any client it communicates with beyond a single request.
- Clients are isolated against changes on the server
- Statelessness promotes redundancy. Clients are not dependent on talking to the same server in two consecutive requests. If a server goes down, just load balance clients to other servers.



HTTP

HTTP IS A STATELESS PROTOCOL, SERVER DOES NOT MAINTAIN ANY STATE FROM ONE REQUEST TO ANOTHER



HTTP STATUS CODES

<https://tools.ietf.org/html/rfc2616#section-10>

	HTTP Status Codes	Description
	1xx	Informational
	2xx	Success
	3xx	Redirection
	4xx	Client Error
	5xx	Server Error



ERRORS

- HTTP has a well-defined set of status codes for responses, some of which indicate an error.
- The status codes are grouped in categories.
- Client does not necessarily need to handle each and every individual code.
- An error can be augmented with more details about what caused the error simply by including text in the body of the response.
- So when throwing and handling errors, RESTful systems converse using a well-known standards.
- Error handling becomes less about the idiosyncratic preferences of developers.



1xx INFORMATIONAL

100	CONTINUE
101	SWITCHING PROTOCOLS
102	WebDAV Processing

2xx SUCCESS

200	OK
201	CREATED
202	ACCEPTED
203	NON-AUTHORITATIVE INFORMATION
204	NO CONTENT
205	RESET CONTENT
206	PARTIAL CONTENT
207	MULTI-STATUS(WebDAV)
208	ALREADY REPORTED (WebDAV)
226	IM USED



3XX REDIRECTION

300	MULTIPLE CHOICES
301	MOVED PERMANENTLY
302	FOUND
303	SEE OTHER
304	NOT MODIFIED
305	USE PROXY
306	UNUSED
307	TEMPORARY REDIRECT
308	PERMANENT REDIRECT (EXPERIMENT)

4XX 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 TIMEDOUT
409	CONFLICT
410	GONE
411	LENGTH REQUIRED



4XX CLIENT ERROR

412	PRECONDITION FAILED
413	REQUEST ENTITY TOO LARGE
414	REQUEST-URI TOO LONG
415	UNSUPPORTED MEDIA TYPE
416	REQUESTED RANGE NOT SATISFIABLE
417	EXPECTATION FAILED
418	RFC 2324
420	ENAHNCE YOUR CLAIM (TWITTER)
422	UNPROCESSABLE ENTITY(WebDAV)
423	LOCKED (WebDAV)
424	Failed Dependency (WebDAV)
425	Reserved for WebDAV

4XX CLIENT ERROR

426	UPGRADE REQUIRED
428	FAILED DEPENDENCY (WebDAV)
429	TOO MANY REQUESTS
431	REQUEST HEADER FIELDS TOO LARGE
444	NO RESPONSE (nginx)
449	RETRY WITH MICROSOFT
450	BLOCKED BY WINDOWS PARENTAL CONTROLS(MICROSOFT)
451	UNAVAILABLE FOR LEGAL REASONS
499	CLIENT CLOSED REQUEST (nginx)



5XX SERVER ERROR



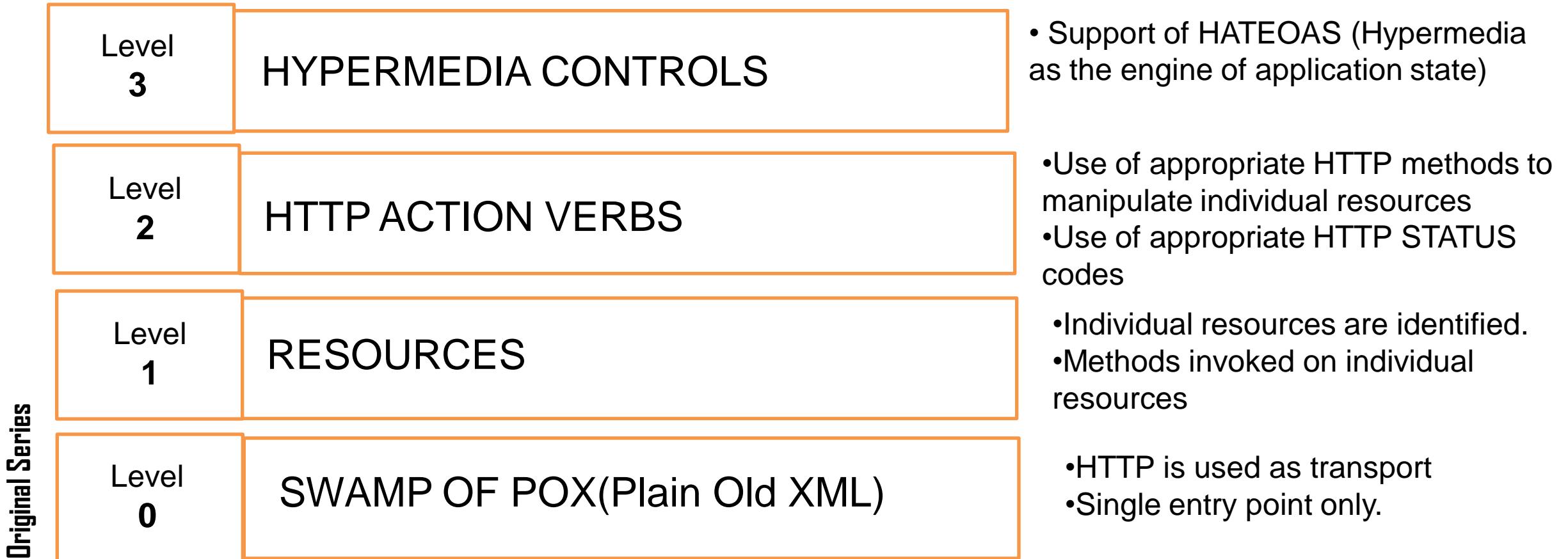
500	INTERNAL SERVER ERROR
501	NOT IMPLEMENTED
502	BAD GATEWAY
503	SERVICE UNAVAILABLE
504	GATEWAY TIMEOUT
505	HTTP VERSION NOT SUPPORTED
506	VARIANT ALSO NEGOTIATES
507	INSUFFICIENT STORAGE (WebDAV)
508	LOOP DETECTED (WebDAV)
509	BANDWIDTH LIMIT EXCEEDED
510	NOT EXTENDED
511	NETWORK AUTHENTICATION REQUIRED
598	NETWORK READ TIMEDOUT ERROR
599	NETWORK CONNECT TIMEOUT ERROR

REST PROPERTIES

- Heterogeneity
- Scalability
- Evolvability
- Visibility
- Reliability
- Efficiency
- Performance
- Manageability
- REST is not a way to call methods over a network without the overhead of SOAP AND WSDL



REST MATURITY MODEL (RICHARDSON MARTURITY MODEL)



CONCERNS

1. RELIABILITY OF NETWORK
2. LATENCY
3. BANDWIDTH
4. SECURITY
5. NETWORK TOPOLOGY
6. TRANSPORTATION COST
7. HETEROGENEOUS NETWORK
8. COMPLEXITY
9. EASY OF MANAGEMENT



ELEMENTS OF RESTFUL ARCHITECTURES

1. COMPONENTS AND CONNECTORS
2. RESOURCES
3. REPRESENTATIONS
4. CONTROL DATA
5. HYPERMEDIA



COMPONENTS

- The processors of resource requests and representations in an application.
- They are categorized by their role
 - Origin server
 - User agent
 - Gateway
 - Proxy

CONNECTORS

- Represent the activities involved in accessing resources and transferring representations.
- Roles provide an interface for components to implement
 - Type of connectors
 - Client
 - Server
 - Cache
 - Resolver
 - tunnel



THE UNIFORM INTERFACE

- Helps in identification of resources uniquely
- Manipulation through representations
- They are self-descriptive messages
- Use hypermedia as the engine of the application state (HATEOAS)

RESOURCE

- Can be any named information or concept
- Maps a named concept to a set of entities over time
- Many to many relationship between named concepts and entities.
- The relationship may be consistent over time or changes dynamically.

ADDRESSES:

ACTION BASED URI: <http://sycliq.com/crop/weatherLookup.do?zipcode=5600039>

RESOURCE BASED URI: <http://sycliq.com/crop/weather/zipcode/5600039>





RESOURCE IDENTIFIER

- Identifies a resource in a component interaction.
- The server is responsible for ensuring that the mapping semantics between identifier and entities do not change.

RESOURCE METADATA

- It is the data that describes the resource
- Metadata could be
 - Location identifier of the resource
 - Identify alternate resource identifiers for different format requests
 - Entity tag information

REPRESENTATION

- Resource state at a point in time
- Can be any sequence of bytes
- A resource can have multiple available representations
- Content negotiation is the process of selecting the best representation of a resource.

REPRESENTATION METADATA

- It is the data that describes the representation.
- It helps the client or server know how to process the bytes
 - How it is structured
 - What it means.





CONTROL DATA

- Data that describes the message sent between components.
- It provides the semantics for the message exchange
- It enables overriding the default behaviour of connectors.

HYPERMEDIA

- Hypermedia is the way to initiate state transitions
- This is used to reduce coupling between the client and server.
- Coupling is reduced by reducing the number of URLs that the client needs to know about.

URIs:Design

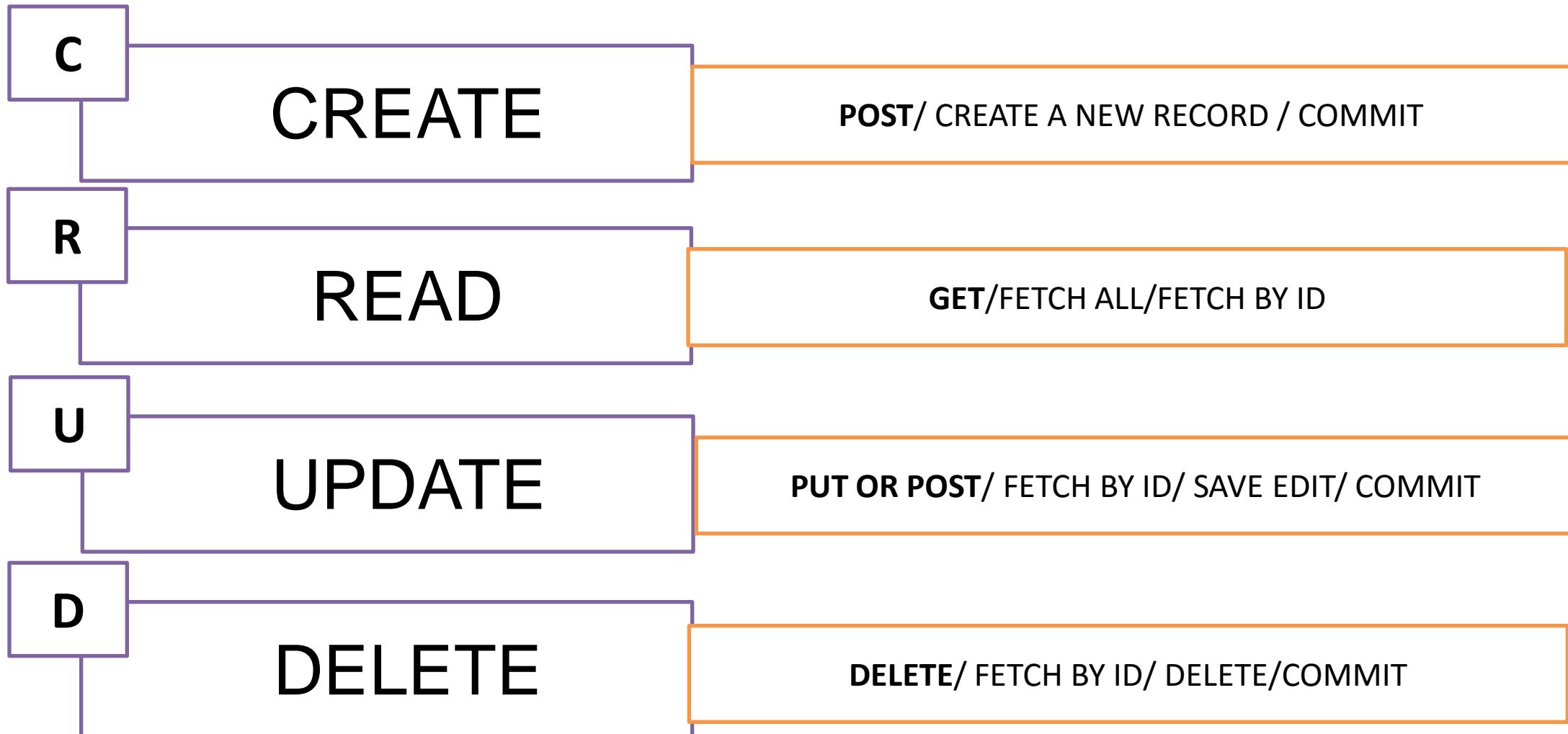
Customers	/customers/{customerId}
Feedback	/customers/{customerId}/feedbacks/{feedbackId}
Filtering Results	
Customers by list	/customers?offset=20&limit=10
Customers by country	/customers?country="usa"
	/customers?country="usa"&offset=20&limit=10

Resource URIs

- **Instance Resource URIs**
- **Collection Resource URIs**
- **Query parameters for pagination and filtering collections**



Mapping CRUD to RMM



MULTIPLE DATA FORMATS

XML

- STRUCTURED AND STRICTLY ADHERED TO A SCHEMA
- EASIER TO VALIDATE
- STRUCTURED REPRESENTATION OF DATA
- REQUIRES XSLT TO PARSE THROUGH THE DATA
- HEAVY
- OFTEN GETS ASSOCIATED WITH SOAP
- **@Produces(MediaType.APPLICATION_XML)**

JSON

- DYNAMIC SCHEMA CHANGES
- KEY / VALUE PAIR
- LOOSER WAY TO REPRESENT DATA
- FLEXIBLE AND EASY TO WORK WITH
- SOMETIMES DIFFICULT TO VALIDATE
- LIGHT WEIGHT
- **@Produces(MediaType.APPLICATION_JSON)**

BINARY/OCTET STREAM

- NOT NECESSARILY AN ALTERNATIVE TO JSON OR XML
- USED TO SERVE OBJECTS
 - FILES
 - IMAGES
 - PDFs
 - **@Produces(MediaType.APPLICATION_OCTET_STREAM)**





RESTful Web Services in Java.

SYED AWASE

IX(A): JAVA WEB SERVICES: JAX-RS



JAX-RS

- Specification for Java RESTful web services
 - Supports plain old Java Objects (POJO) through URLs.
- CXF support for JAX-RS version
 - JAX-RS 2.0 – CXF 2.x mostly supports
 - CXF 3.x fully supports
 - JAX-RS 1.1 – CXF 2.x+ fully supports
- REST was built on the principles of HTTP
- Services can return
 - XML
 - JSON
 - HTML
 - Plain Text
 - Binary/Octet (Images/PDF)



Implementing JAX-RS

- Available in the **cxf-rt-frontend-jaxrs** library
- Supported by the HTTP transport and CXF Servlet
- Configured through Spring application context

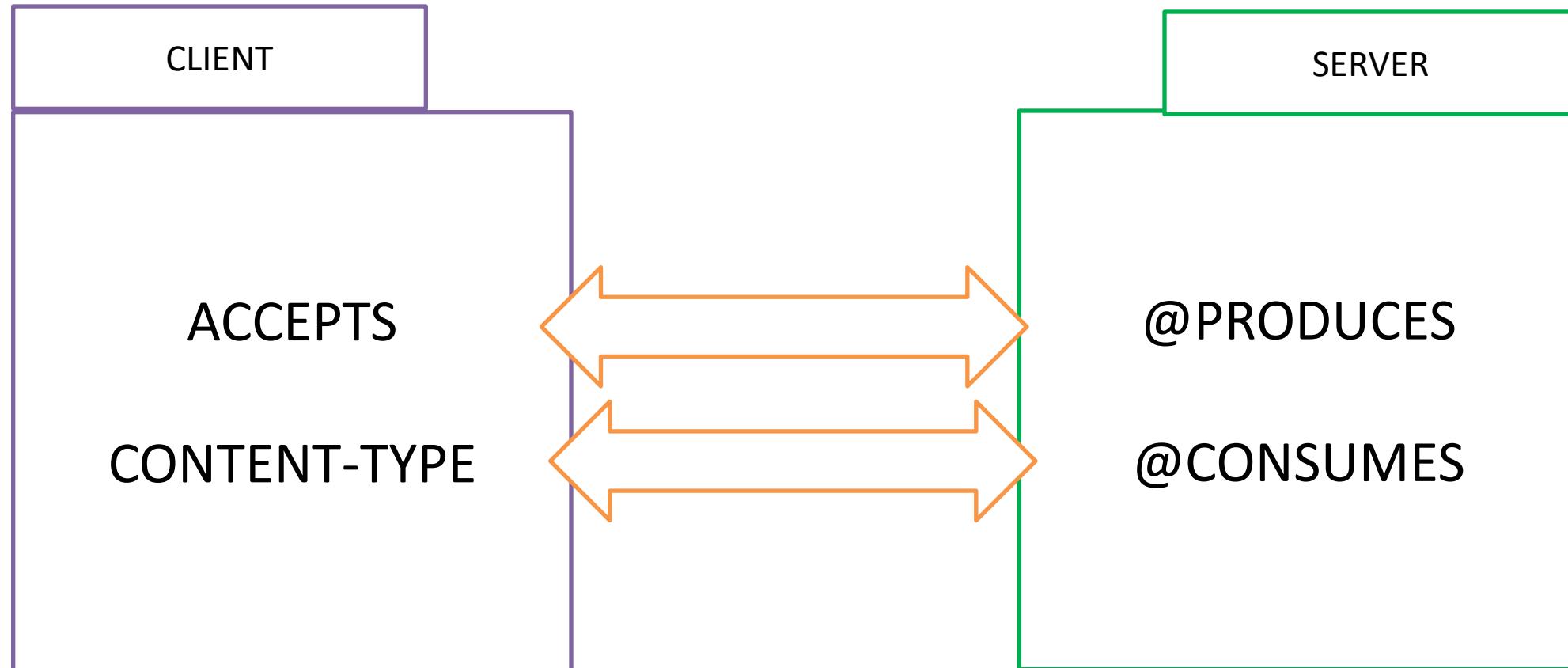


JAX-RS Annotations

Annotation	Package Detail/Import Statement
@GET	Import javax.ws.rs.GET
@Produces	Import javax.ws.rs.Produces
@Path	Import javax.ws.rs.Path
@PathParam	Import javax.ws.rsPathParam
@QueryParam	Import javax.ws.rsQueryParam
@POST	Import javax.ws.rs.POST
@Consumes	Import javax.ws.rs.Consumes
@FormParam	Import javax.ws.rs.FormParam
@PUT	Import javax.ws.rs.PUT
@DELETE	Import javax.ws.rs.DELETE
@MatrixParam	



CONTENT NEGOTIATION



JERSEY

IX(B): PLAY BOOK: CREATING A BASIC RESTFUL SERVICE

Maven Installation of Jersey

- Maven Archetype Installation of Jersey

```
>mvn archetype:generate -  
DarchetypeGroupId=org.glassfish.jersey.archet  
ypes -DarchetypeArtifactId=jersey-  
quickstart-webapp -DarchetypeVersion=2.2
```



Step 1

- Create maven project
 - Archetype=>
 - Jersey-quickstart-webapp

The screenshot shows the 'New Maven Project' dialog from an IDE. The title bar says 'New Maven Project'. The main area is titled 'New Maven project' and 'Select an Archetype'. A catalog dropdown is set to 'All Catalogs' and a filter input field contains 'jersey'. Below is a table of available archetypes:

Group Id	Artifact Id	Version
org.fusesource.scalate.tooling	scalate-archetype-jersey_2.10	1.6.1
org.fusesource.scalate.tooling	scalate-archetype-jersey_2.9	1.6.1
org.glassfish.jersey.archetypes	jersey-example-java8-webapp	2.26
org.glassfish.jersey.archetypes	jersey-heroku-webapp	2.26
org.glassfish.jersey.archetypes	jersey-quickstart-grizzly2	2.26
org.glassfish.jersey.archetypes	jersey-quickstart-webapp	2.26
org.scalatra.scalate.tooling	scalate-archetype-jersey_2.10	1.7.1

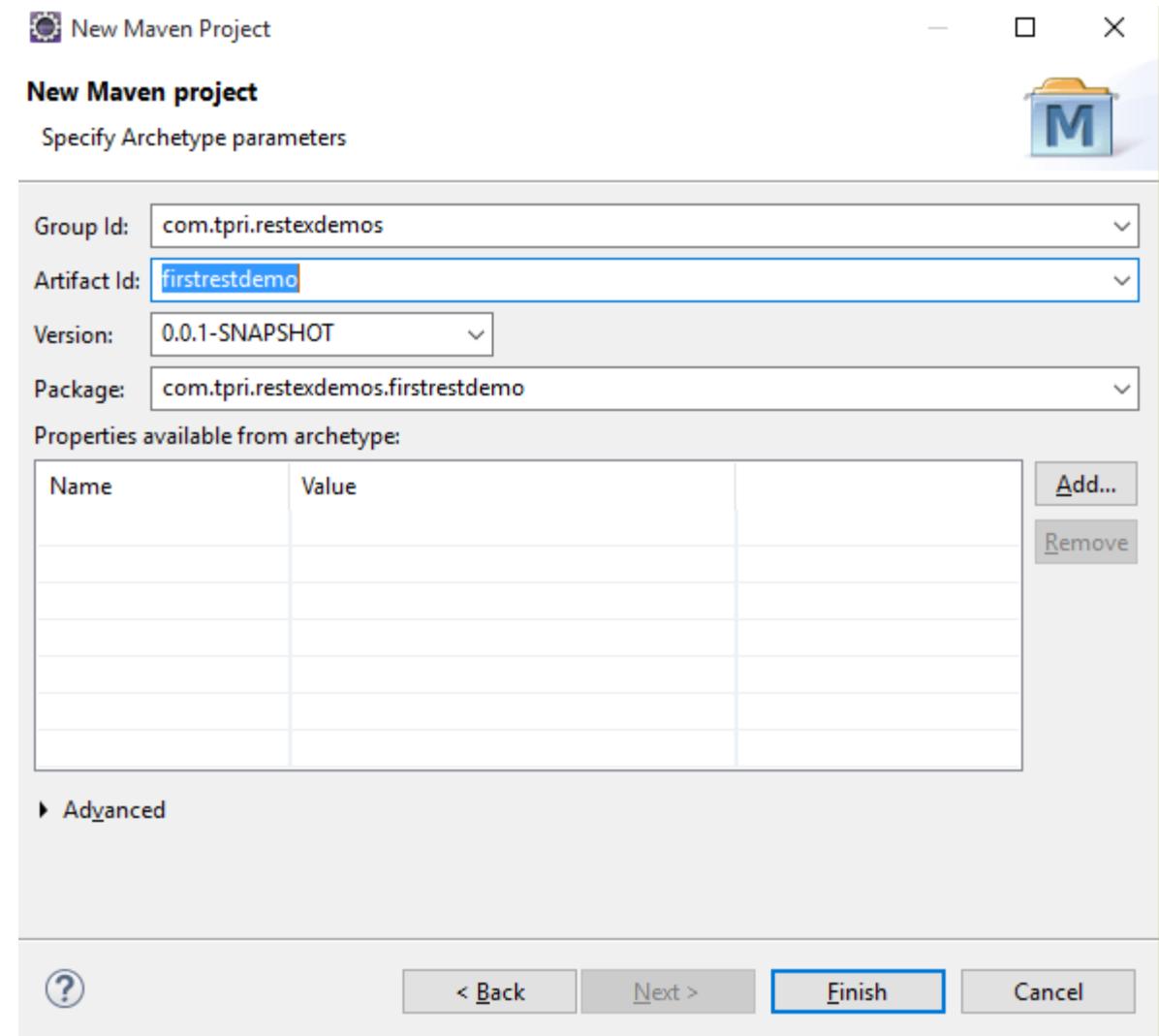
An info box below the table states: 'An archetype which contains a quick start Jersey-based web application project.' with a link 'http://repo1.maven.org/maven2'. At the bottom are checkboxes for 'Show the last version of Archetype only' and 'Include snapshot archetypes', and a 'Add Archetype...' button. Navigation buttons at the bottom include '?', '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted.

<https://mvnrepository.com/>



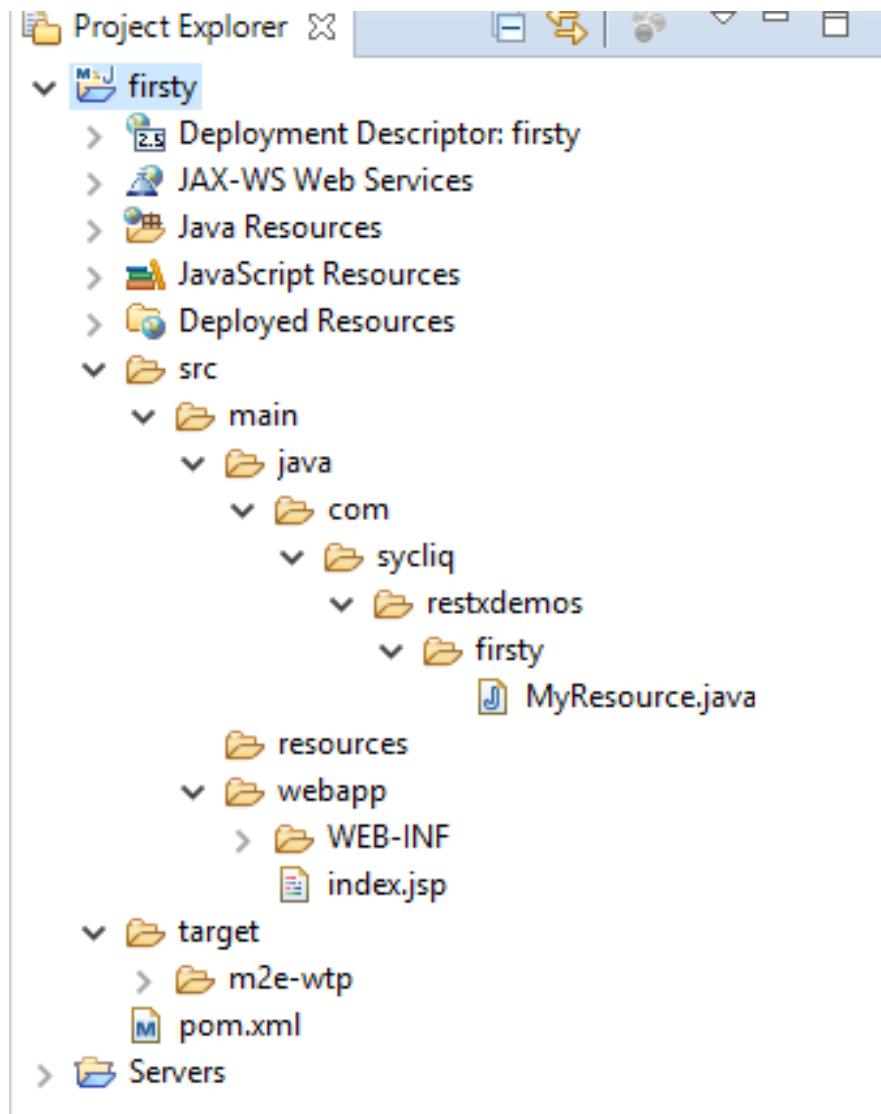
Step 2:

- define groupId, artifactId



Step 3

- MAVEN downloads and scaffolds an restful web-app of archetype jersey.



Step 4

- Maven scaffolds and includes the dependencies for successfully rendering a restful endpoint

- ✓  Maven Dependencies
 - >  jersey-container-servlet-core-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-container-servlet\2.21\jersey-container-servlet-core-2.21.jar
 - >  javax.inject-2.4.0-b31.jar - C:\Users\SyedAwase\.m2\repository\org\javax\inject\2.4.0-b31\javax.inject-2.4.0-b31.jar
 - >  jersey-common-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-common\2.21\jersey-common-2.21.jar
 - >  javax.annotation-api-1.2.jar - C:\Users\SyedAwase\.m2\repository\org\javax\annotation\api\1.2\javax.annotation-api-1.2.jar
 - >  jersey-guava-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-guava\2.21\jersey-guava-2.21.jar
 - >  hk2-api-2.4.0-b31.jar - C:\Users\SyedAwase\.m2\repository\org\hk2\hk2-api\2.4.0-b31\hk2-api-2.4.0-b31.jar
 - >  hk2-utils-2.4.0-b31.jar - C:\Users\SyedAwase\.m2\repository\org\hk2\hk2-utils\2.4.0-b31\hk2-utils-2.4.0-b31.jar
 - >  aopalliance-repackaged-2.4.0-b31.jar - C:\Users\SyedAwase\.m2\repository\org\aopalliance\repackaged\2.4.0-b31\repackaged-2.4.0-b31.jar
 - >  hk2-locator-2.4.0-b31.jar - C:\Users\SyedAwase\.m2\repository\org\hk2\hk2-locator\2.4.0-b31\hk2-locator-2.4.0-b31.jar
 - >  javassist-3.18.1-GA.jar - C:\Users\SyedAwase\.m2\repository\org\javassist\javassist\3.18.1-GA\javassist-3.18.1-GA.jar
 - >  osgi-resource-locator-1.0.1.jar - C:\Users\SyedAwase\.m2\repository\org\osgi\osgi-resource-locator\1.0.1\osgi-resource-locator-1.0.1.jar
 - >  jersey-server-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-server\2.21\jersey-server-2.21.jar
 - >  jersey-client-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-client\2.21\jersey-client-2.21.jar
 - >  jersey-media-jaxb-2.21.jar - C:\Users\SyedAwase\.m2\repository\org\jersey-project\jersey-media-jaxb\2.21\jersey-media-jaxb-2.21.jar
 - >  validation-api-1.1.0.Final.jar - C:\Users\SyedAwase\.m2\repository\org\hibernate\Validation\validation-api\1.1.0.Final\validation-api-1.1.0.Final.jar
 - >  javax.ws.rs-api-2.0.1.jar - C:\Users\SyedAwase\.m2\repository\javax\ws\rs\api\2.0.1\javax.ws.rs-api-2.0.1.jar
 - > ...



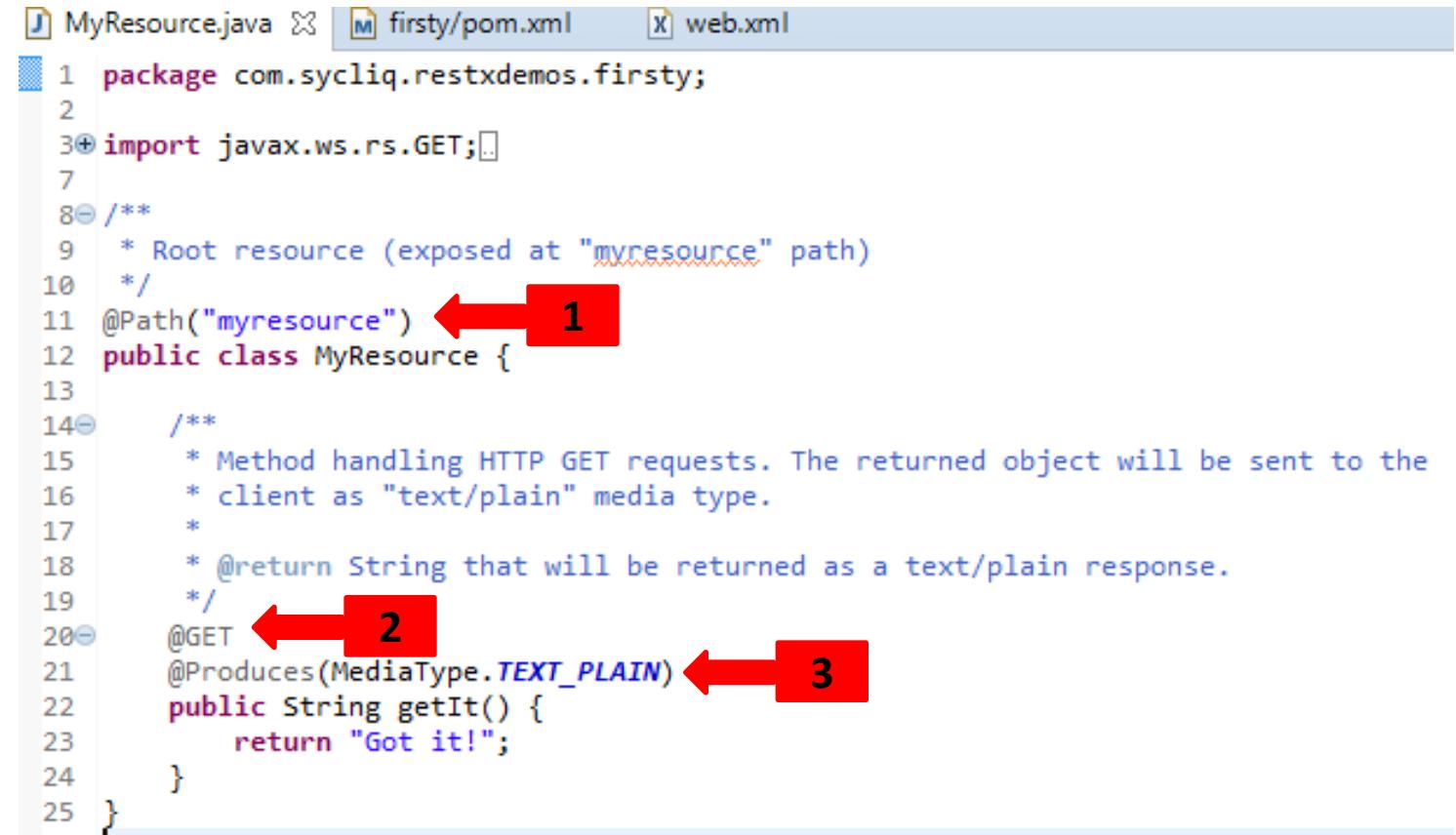
Web.xml

```
MyResource.java    firstly/pom.xml    web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- This web.xml file is not required when using Servlet 3.0 container,
3      see implementation details http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->
4 <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
7 <servlet>
8   <servlet-name>Jersey Web Application</servlet-name>
9   <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
10 <init-param>
11   <param-name>jersey.config.server.provider.packages</param-name>
12   <param-value>com.syqliq.restxdemos.firsty</param-value>
13 </init-param>
14 <load-on-startup>1</load-on-startup>
15 </servlet>
16 <servlet-mapping>
17   <servlet-name>Jersey Web Application</servlet-name>
18   <url-pattern>/webapi/*</url-pattern>
19 </servlet-mapping>
20 </web-app>
```



Step 5

1. Creating a service using the following steps
 - a. @Path- the path to the service
 - b. @GET-the request method
 - c. @Produces – the response type from the Service.
 - @Produces can take an Array of multiple types.



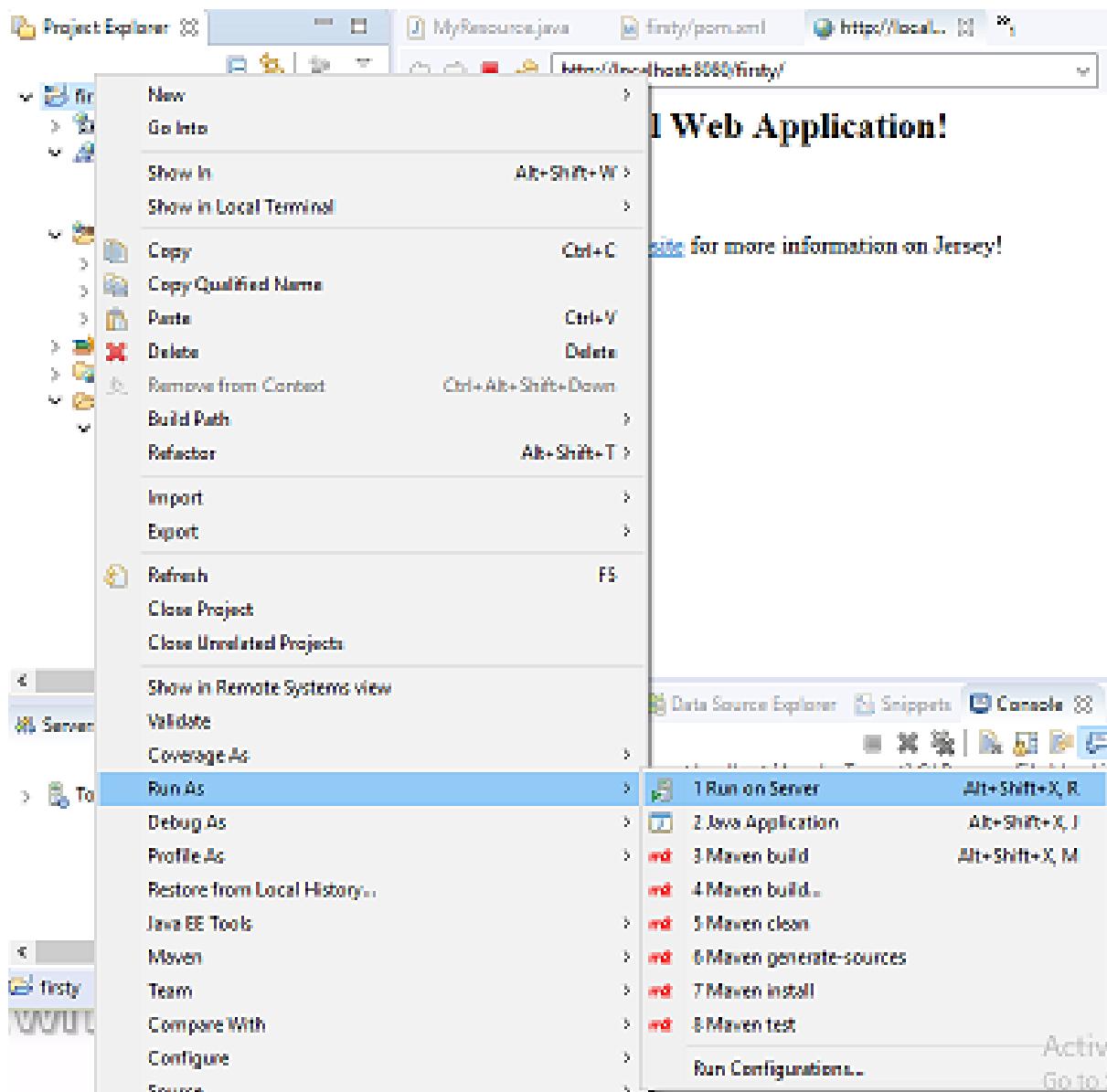
```
MyResource.java  firsty/pom.xml  web.xml
1 package com.sycliq.restxdemos.firsty;
2
3 import javax.ws.rs.GET;
4
5 /**
6  * Root resource (exposed at "myresource" path)
7  */
8
9 @Path("myresource") ← 1
10
11 public class MyResource {
12
13
14 /**
15  * Method handling HTTP GET requests. The returned object will be sent to the
16  * client as "text/plain" media type.
17  *
18  * @return String that will be returned as a text/plain response.
19  */
20 @GET ← 2
21 @Produces(MediaType.TEXT_PLAIN) ← 3
22 public String getIt() {
23     return "Got it!";
24 }
25 }
```





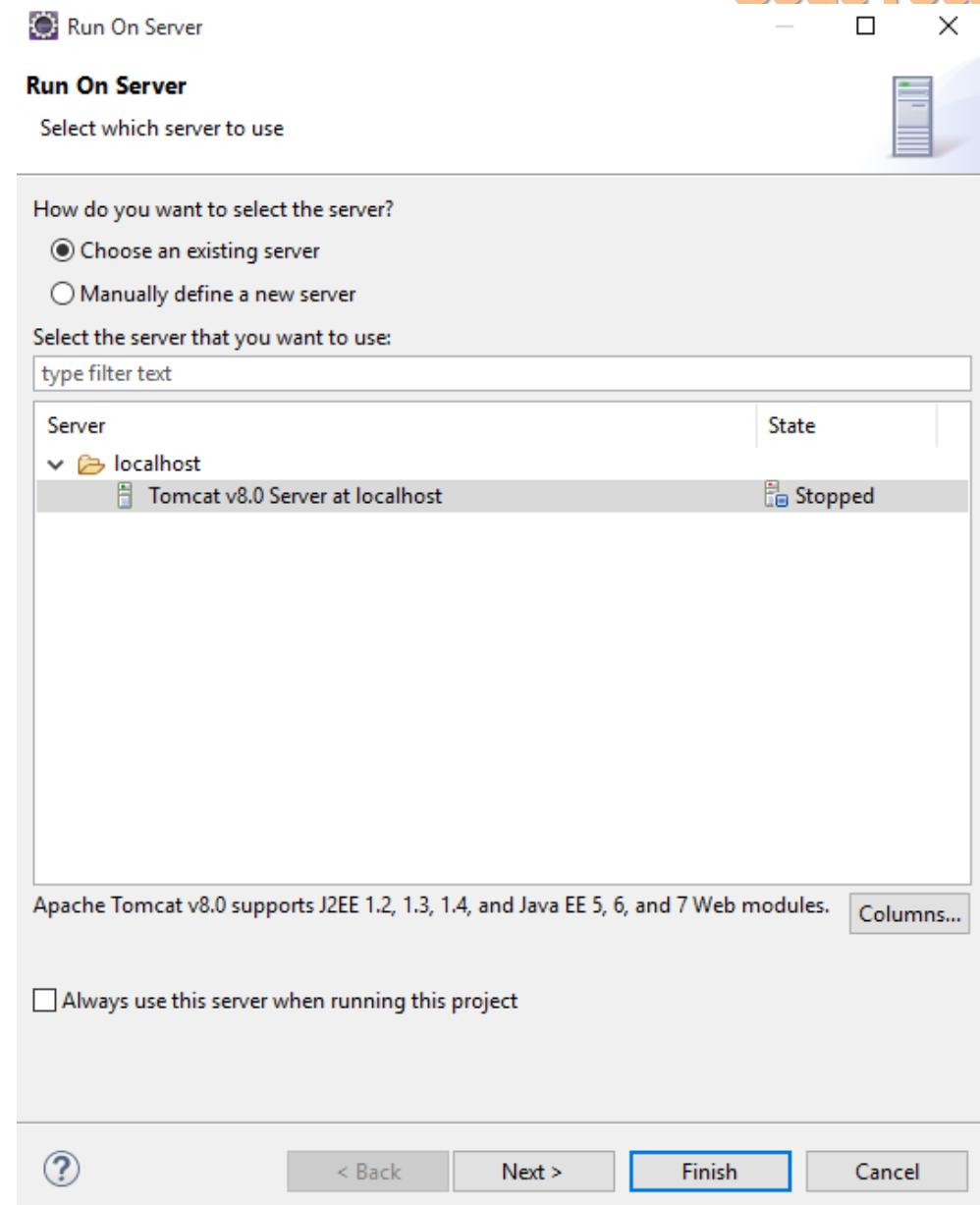
Step 6

- run your jersey webapp quickstart project on the server.



Step 7

- select appropriate Tomcat Server to render your restful endpoint.



Run On Server**Add and Remove**

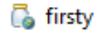
Modify the resources that are configured on the server



Move resources to the right to configure them on the server

Available:

Configured:



firsty

Add >**< Remove****Add All >>****<< Remove All**

< Back

Next >

Finish

Cancel

Step 8

- choose the application to render



The screenshot shows a web browser window with two tabs. The active tab is titled 'MyResource.java' and has the URL 'http://localhost:8080/firsty/'. The content of the page is a large bold heading 'Jersey RESTful Web Application!' followed by the text 'Jersey resource' and 'Visit [Project Jersey website](#) for more information on Jersey!'. The browser interface includes standard buttons for back, forward, and search.

Step 9

- Your Jersey RESTful web application is up and running at
<http://localhost:8080/firsty>



Step 10

- Navigate to the webapi resource to view the rendering of the endpoint
- [http://localhost:8080/firs
ty/webapi/myresource](http://localhost:8080/firsty/webapi/myresource)



JERSEY

IX(B): PLAY BOOK: GET COMMUTER SERVICE

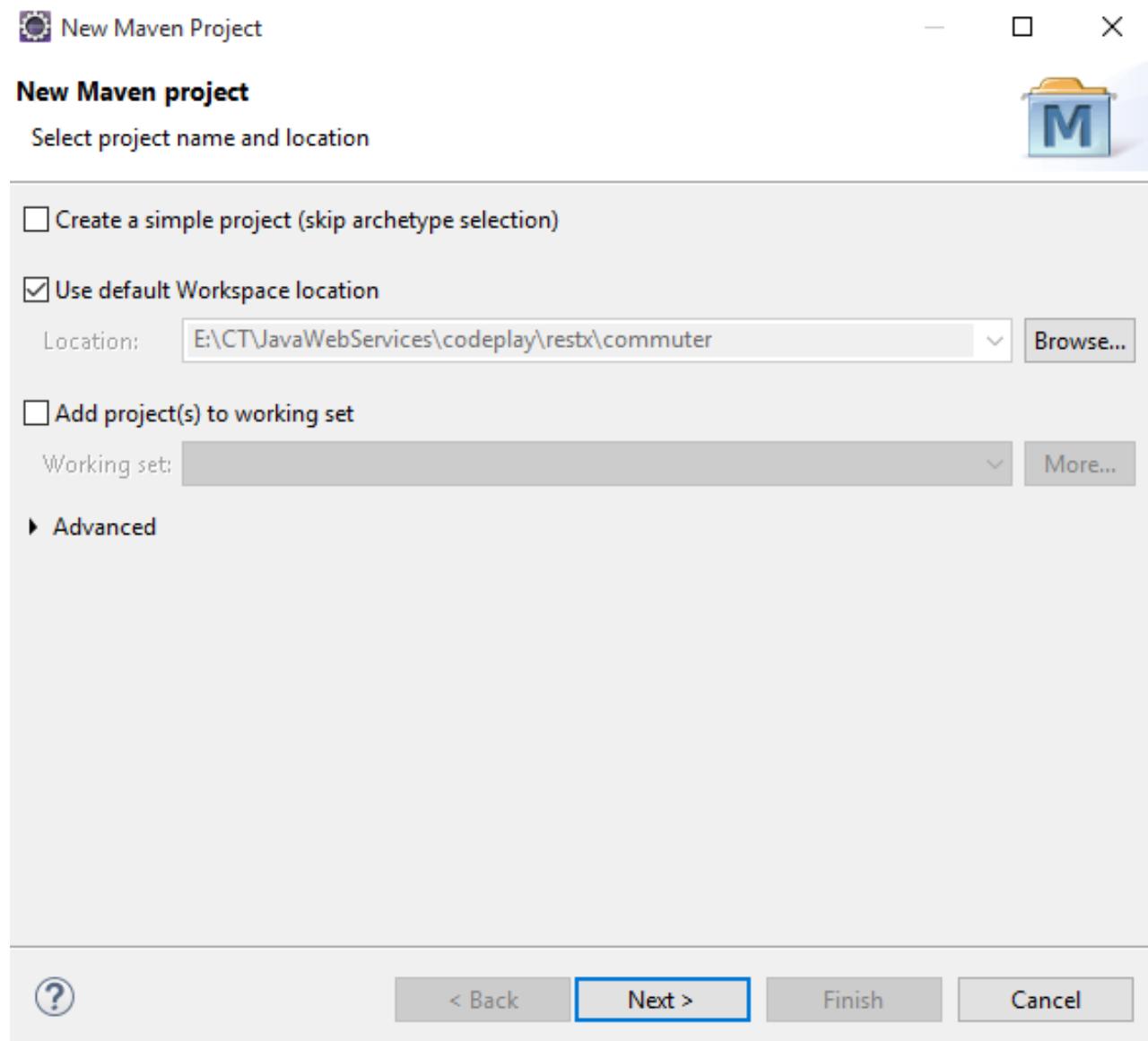
@Produces(MediaType.APPLICATION_XML)





Step 1:

- create a Maven Project



Step 2:

- Maven Project of Archetype
 - Org.glassfish.jersey.archetypes
 - Jersey-quickstart-webapp
 - 2.21 version

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0
org.glassfish.jersey.archetypes	jersey-quickstart-webapp	2.21

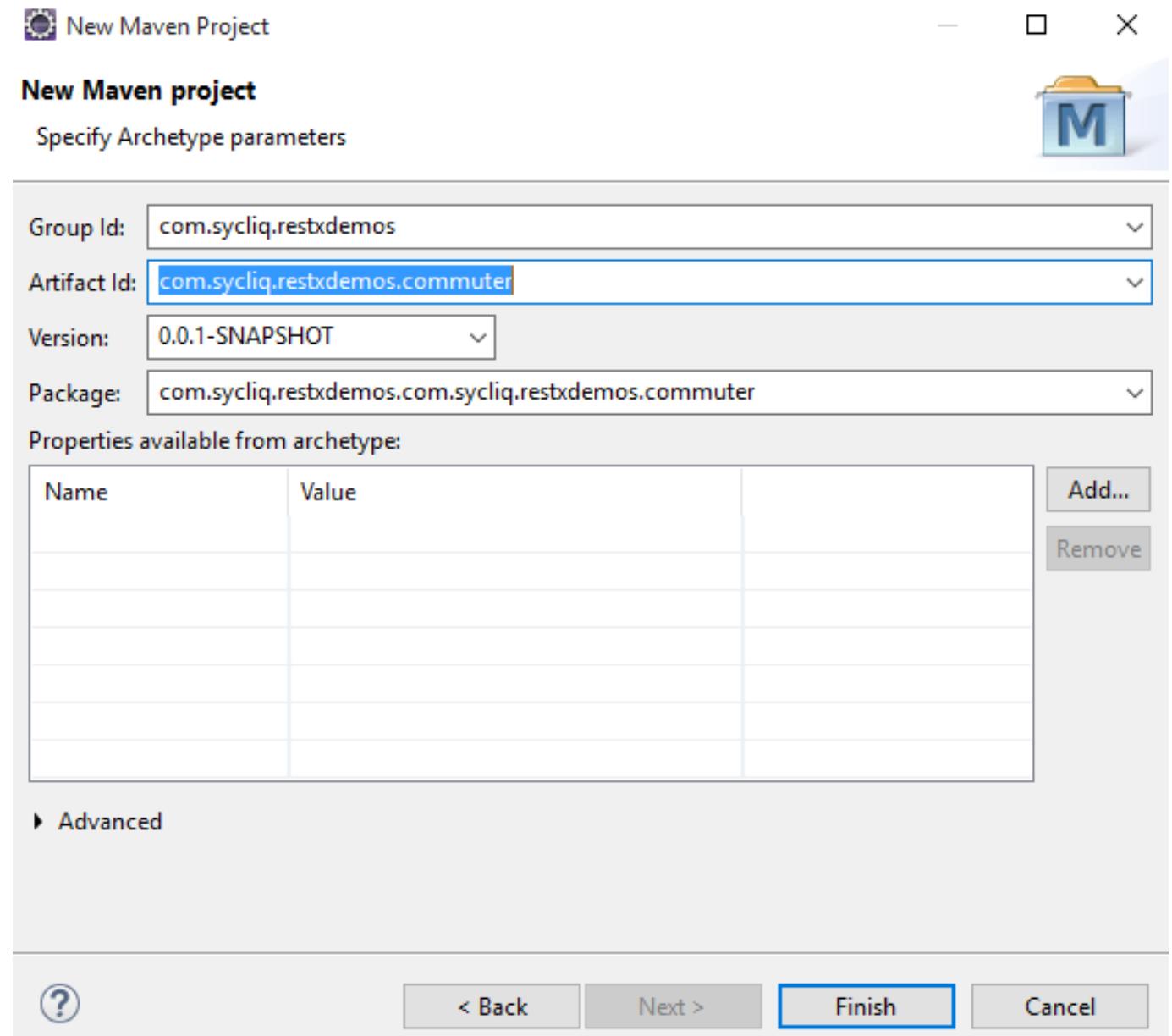
Show the last version of Archetype only Include snapshot archetypes

► Advanced



Step 3:

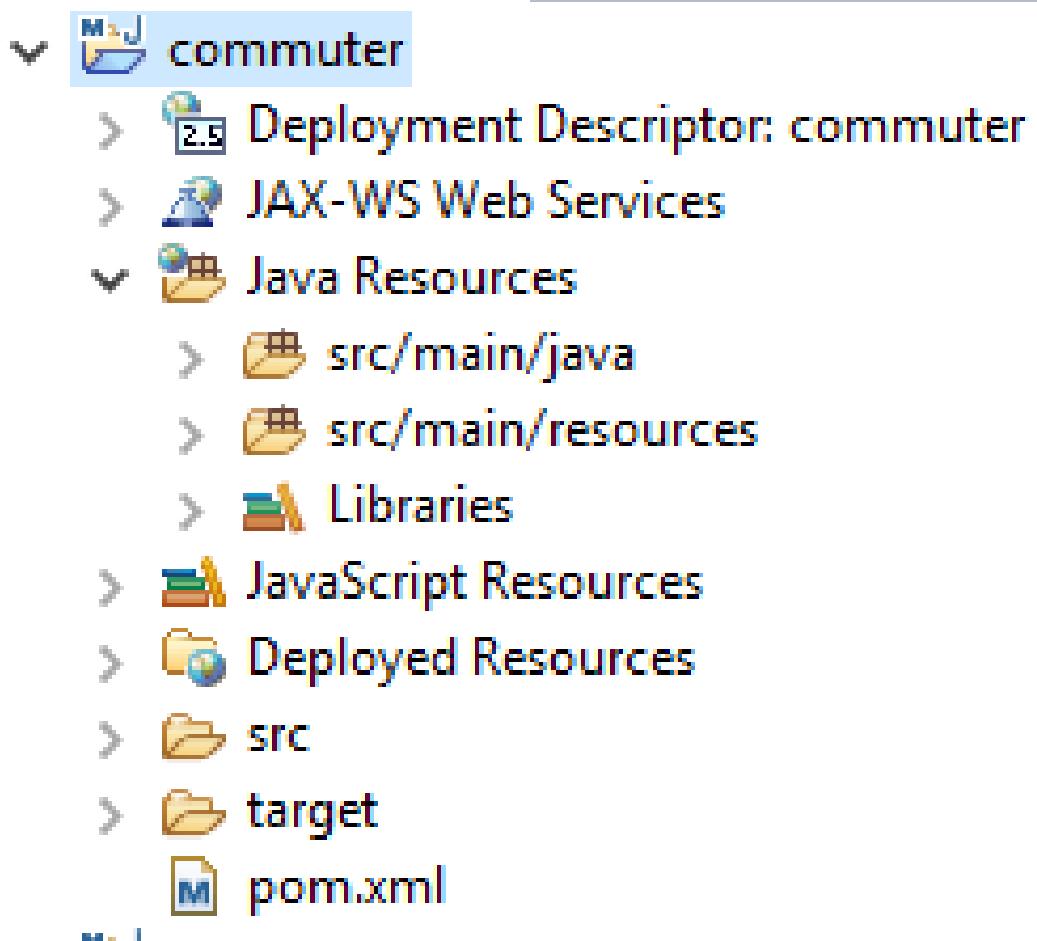
- Create a project with ArtifactId
 - com.syqliq.restxdemos.commuter





Step 4:

- creates a scaffolded jersey webapplication template as shown in the figure.

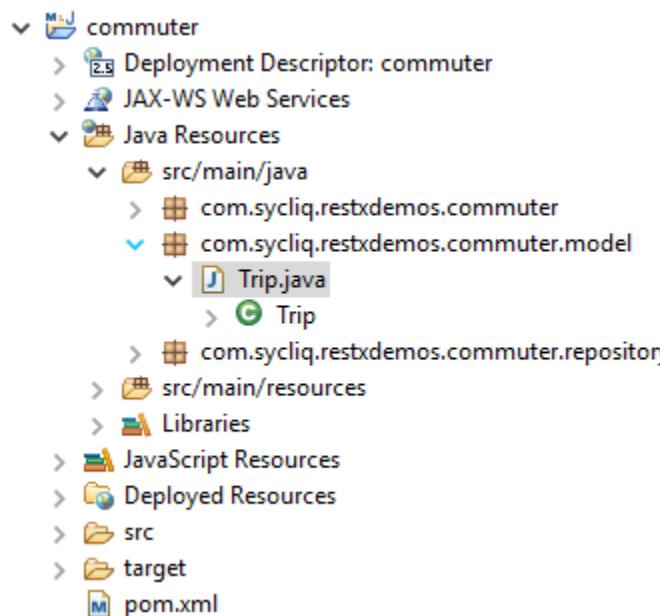


Step 5

- create a package

com.sycliq.restxdemos.co
mmuter.model

- class Trip.java



```
package com.sycliq.restxdemos.commuter.model;  
  
import javax.xml.bind.annotation.XmlRootElement;  
  
@XmlRootElement  
public class Trip {  
    private String tripdescription;  
    private int duration;  
    public String getTripdescription() {  
        return tripdescription;  
    }  
    public void setTripdescription(String tripdescription) {  
        this.tripdescription = tripdescription;  
    }  
    public int getDuration() {  
        return duration;  
    }  
    public void setDuration(int duration) {  
        this.duration = duration;  
    }  
}
```



Step 6

- create a repository

com.sycliq.restxdemos.co
mmuter.repository

- create a stub for
DAO/DAL

- class TripRepositoryStub.java

▼ com.sycliq.restxdemos.commuter.repository
 > TripRepository.java
 > TripRepositoryStub.java

```
import java.util.List;

import com.sycliq.restxdemos.commuter.model.Trip;

//DAL mimic a database
public class TripRepositoryStub implements TripRepository {

    /* (non-Javadoc)
     * @see com.sycliq.restxdemos.commuter.repository.TripRepository#findAllTrips()
     */
    @Override
    public List<Trip> findAllTrips(){
        List<Trip> trips = new ArrayList<Trip>();
        Trip tripOne = new Trip();
        Trip tripTwo = new Trip();

        tripOne.setTripdescription("DownTownTrip");
        tripOne.setDuration(45);

        tripTwo.setTripdescription("Airport");
        tripTwo.setDuration(72);

        trips.add(tripOne);
        trips.add(tripTwo);

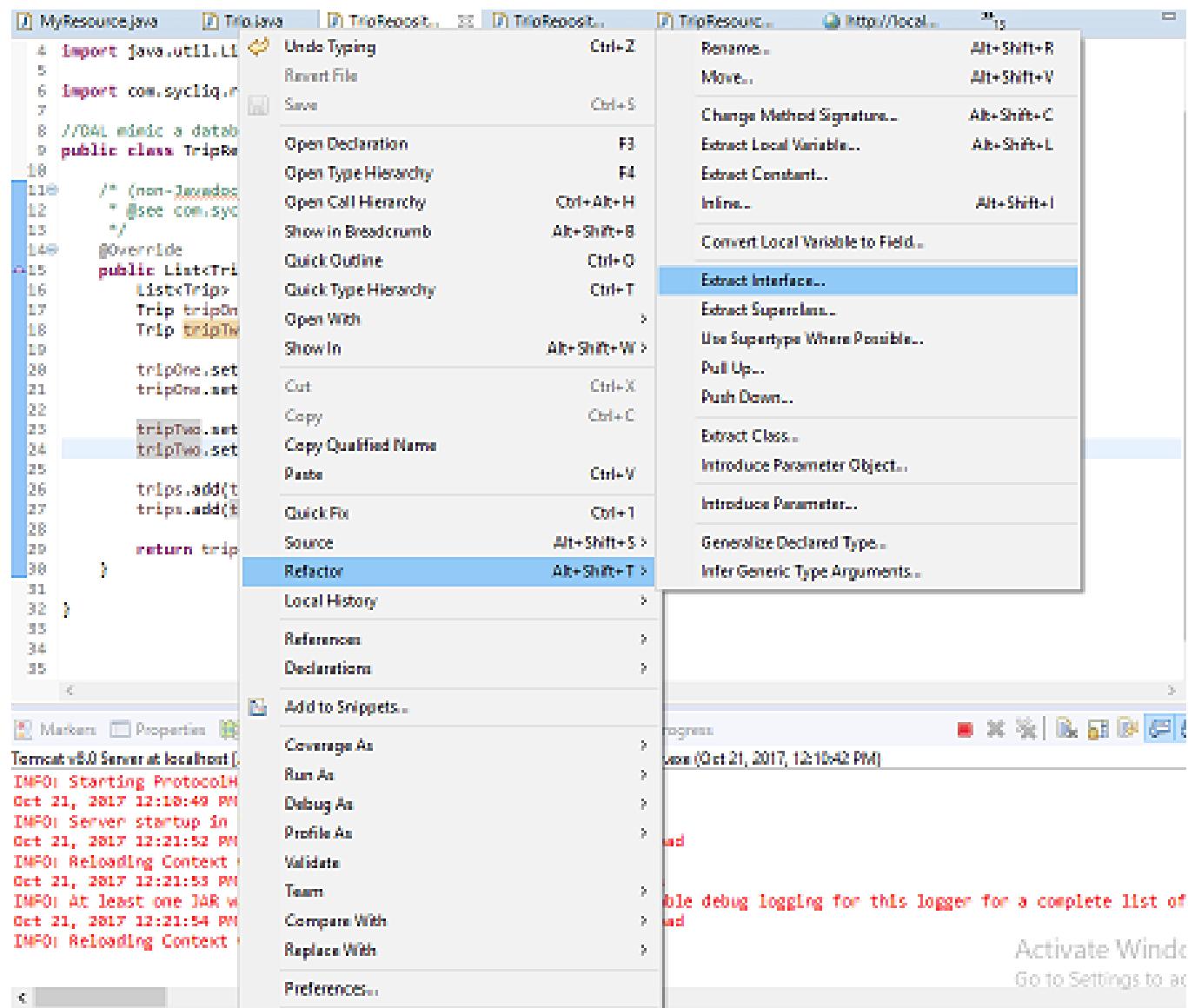
        return trips;
    }
}
```



Step 7

- refactor to generate TripRepository Interface

com.syqliq.restxdemos.commuter.repository
 > TripRepository.java
 > TripRepositoryStub.java





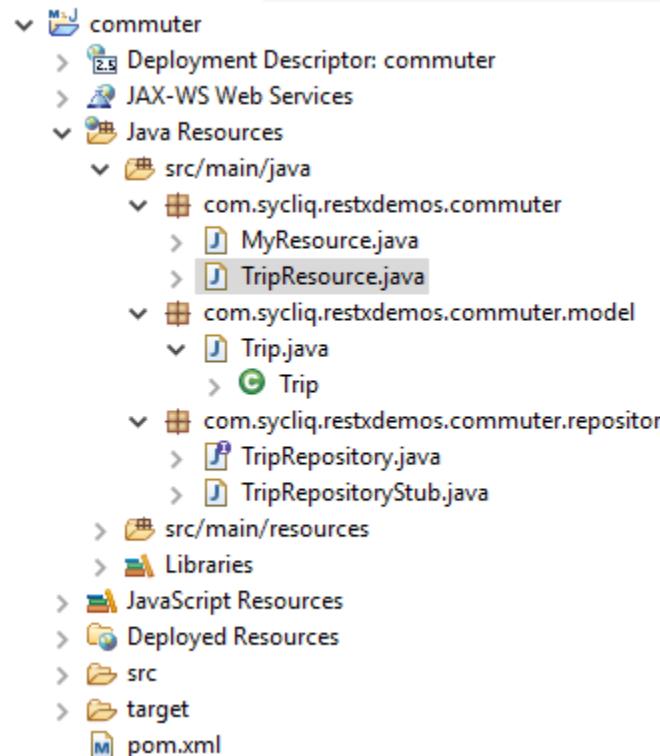
Step 8

- extract the interface from TripRepositoryStub as shown in the figure.

```
package com.sycliq.restxdemos.commuter.repository;  
import java.util.List;  
public interface TripRepository {  
    List<Trip> findAllTrips();  
}
```

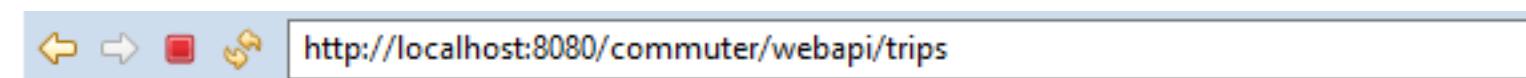
Step 9:

- create TripResource Service to render a endpoint view to display a list of trips.



```
package com.sycliq.restxdemos.commuter;  
import java.util.List;  
  
import javax.ws.rs.GET;  
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;  
  
import com.sycliq.restxdemos.commuter.model.Trip;  
import com.sycliq.restxdemos.commuter.repository.TripRepository;  
import com.sycliq.restxdemos.commuter.repository.TripRepositoryStub;  
  
@Path("trips") ← 1  
public class TripResource {  
  
    private TripRepository tripRepository=new TripRepositoryStub();  
  
    @GET ← 2  
    @Produces(MediaType.APPLICATION_XML) ← 3  
    public List<Trip> getAllTrips() {  
        return tripRepository.findAllTrips();  
    }  
}
```





Step 10:

- Run your application

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <trips>
  - <trip>
    <duration>45</duration>
    <tripdescription>DownTownTrip</tripdescription>
  </trip>
  - <trip>
    <duration>72</duration>
    <tripdescription>Airport</tripdescription>
  </trip>
</trips>
```





JERSEY

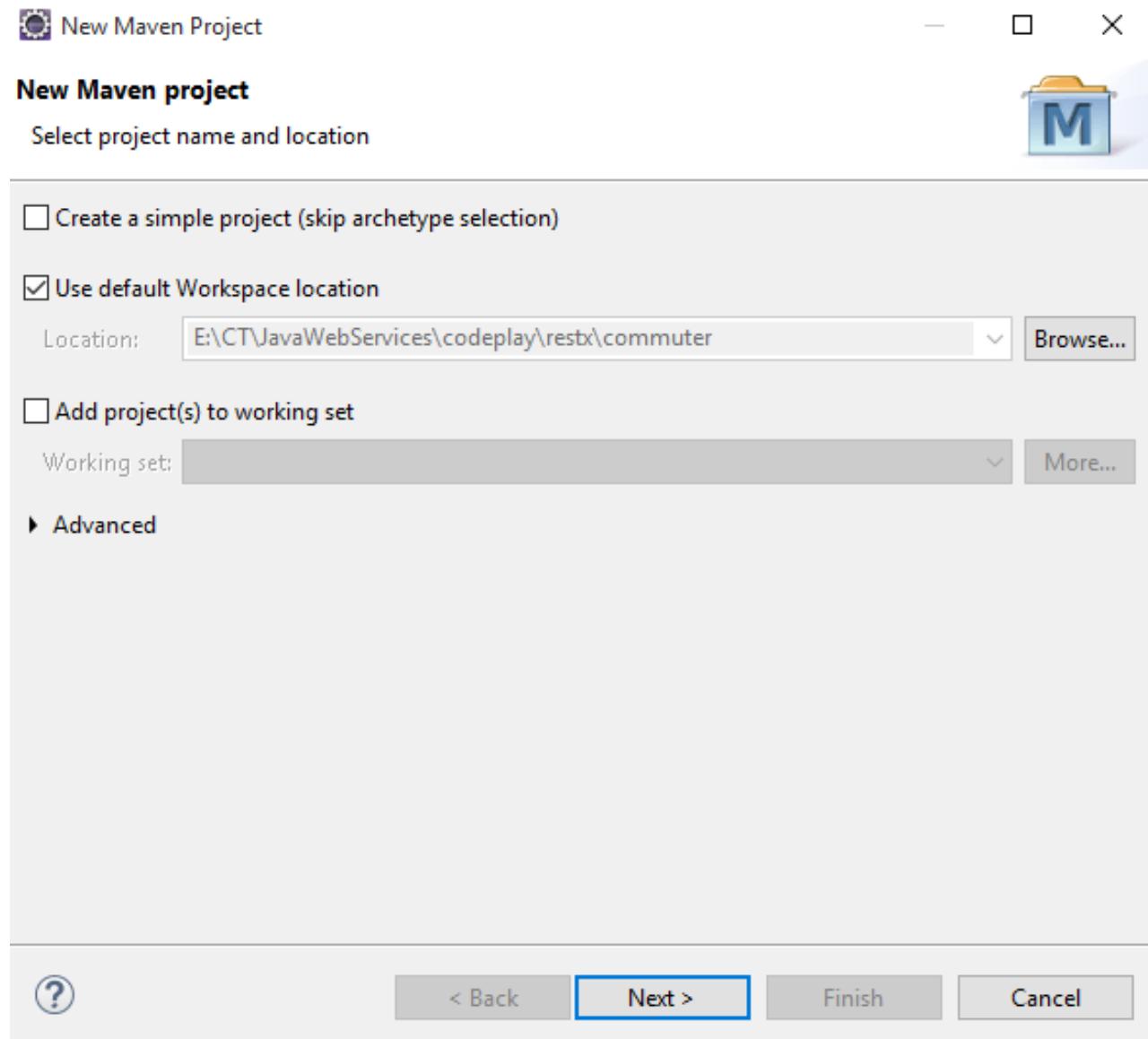
IX(C): PLAY BOOK: GET PRODUCT SERVICE

@Produces(MediaType.APPLICATION_JSON)



Step 1:

- create a Maven Project



Step 2:

- Maven Project of Archetype
 - Org.glassfish.jersey.archetypes
 - Jersey-quickstart-webapp
 - 2.21 version

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

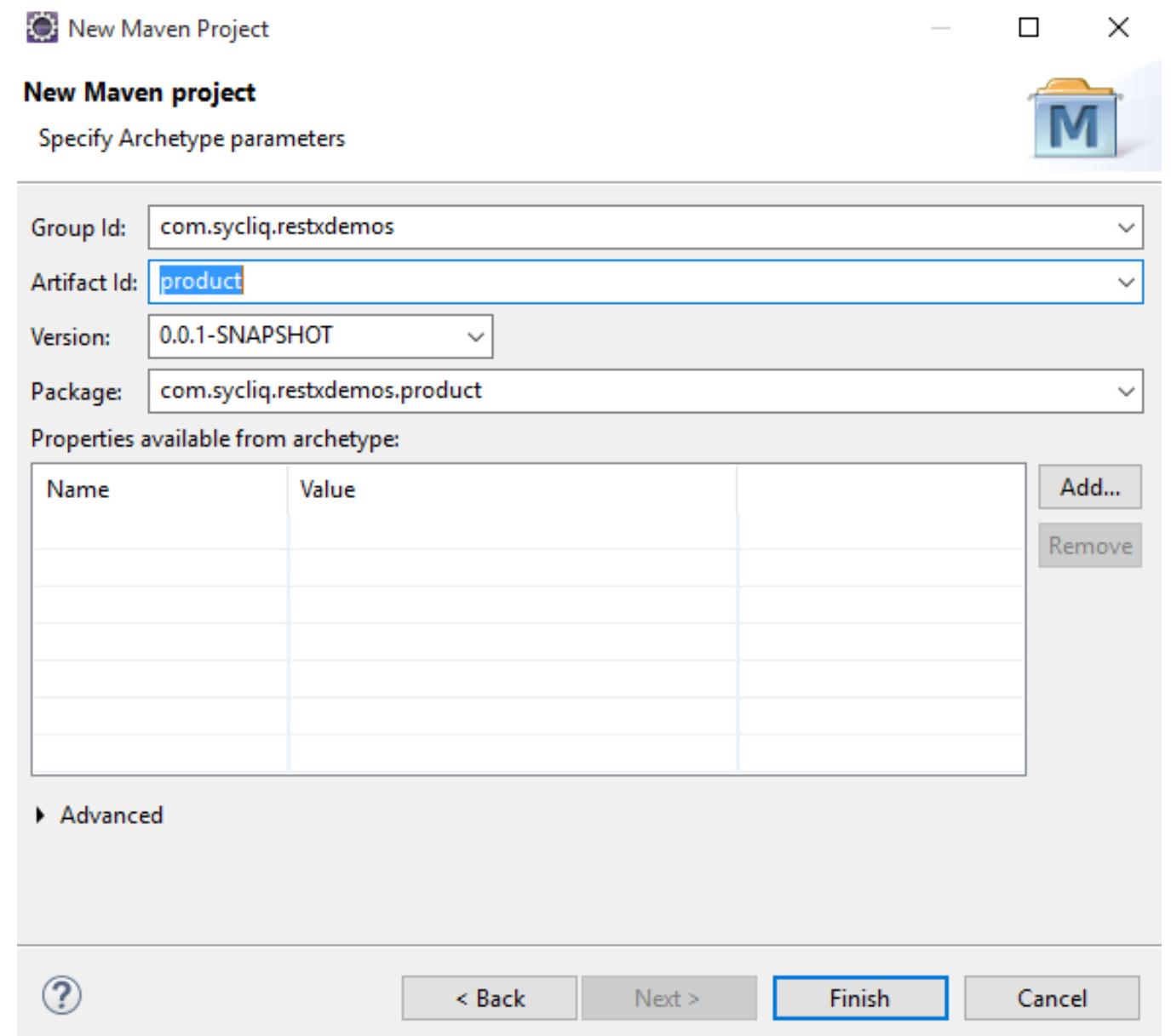
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0
org.glassfish.jersey.archetypes	jersey-quickstart-webapp	2.21

Show the last version of Archetype only Include snapshot archetypes

► Advanced



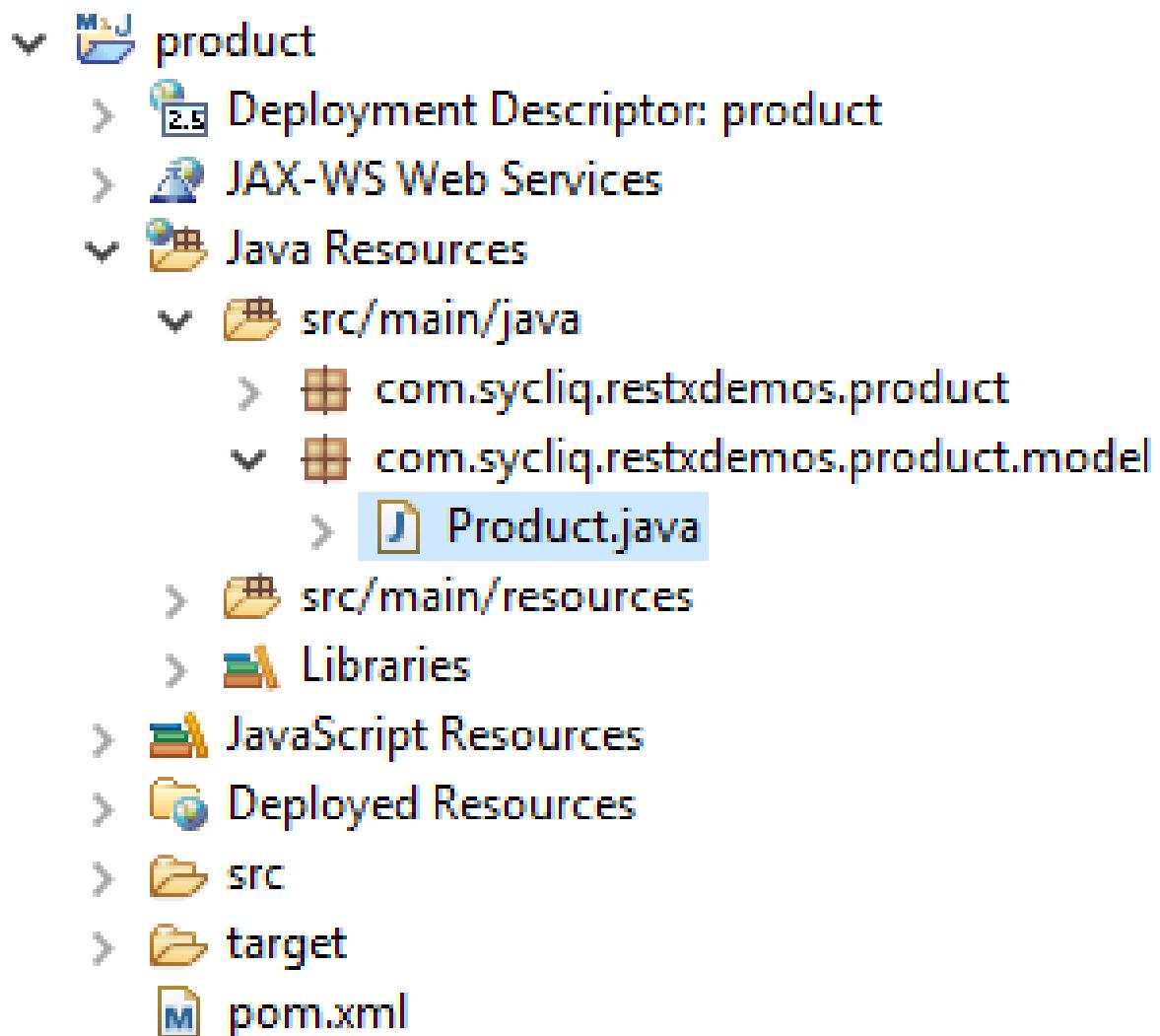
Step 3:





Step 4

- maven archetype jersey project scaffolded template



Step 5:

- create Product model

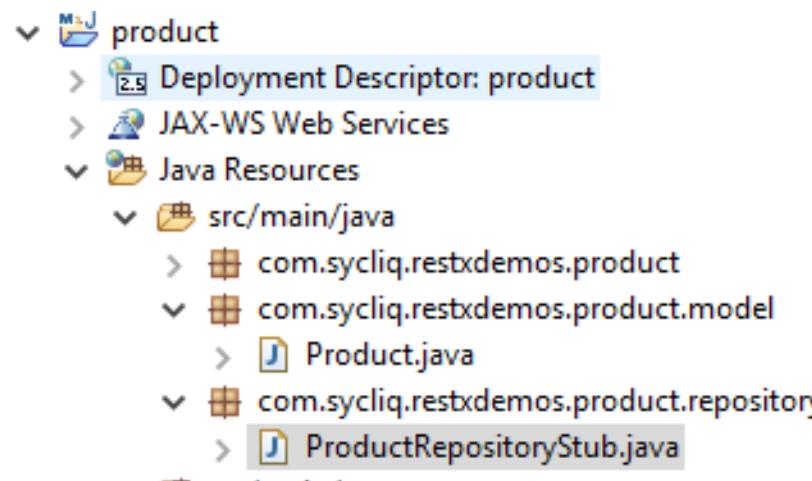
↳ com.sycliq.restdemos.product.model
↳ Product.java

```
J Product.java ✘
7  * @author SyedAwase
8  *
9  */
10 public class Product {
11
12     private String name;
13     private String image;
14     private String description;
15     private String category;
16     private int price;
17     private int quantity;
18     private int shipping;
19     private String location;
20     public String getName() {
21         return name;
22     }
23     public void setName(String name) {
24         this.name = name;
25     }
26     public String getImage() {
27         return image;
28     }
29     public void setImage(String image) {
30         this.image = image;
31     }
32     public String getDescription() {
33         return description;
34     }
35     public void setDescription(String description) {
36         this.description = description;
37     }
38     public String getCategory() {
39         return category;
40     }
41     public void setCategory(String category) {
42         this.category = category;
43     }
44     public int getPrice() {
45         return price;
46     }
47     public void setPrice(int price) {
48         this.price = price;
49     }
50     public int getQuantity() {
51         return quantity;
52     }
53     public void setQuantity(int quantity) {
```



Step 6

- create
ProductRepositoryStub



```
package com.sycliq.restdemos.product.repository;

import java.util.ArrayList;
import java.util.List;

import com.sycliq.restdemos.product.model.Product;

/**
 * @author SyedAwase
 */
public class ProductRepositoryStub {

    public List<Product> findAllProducts(){

        List<Product> products = new ArrayList<Product>();

        Product prodone = new Product();
        Product prodtwo = new Product();

        prodone.setName("OnePlus Two 64GB");
        prodone.setImage("http://i.ebayimg.com/00/s/Mzc4WDcyMA==/z/aUoAAOSwyQtV4-Kg/$_1");
        prodone.setDescription("With the OnePlus 2, we have something bold to say. We be");
        prodone.setCategory("Electronics");
        prodone.setPrice(34999);
        prodone.setQuantity(1);
        prodone.setShipping(100);
        prodone.setLocation("Punjab");

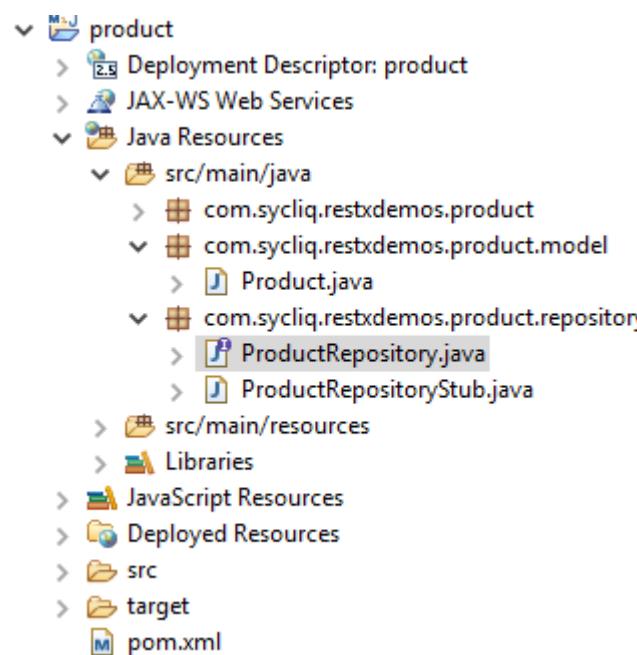
        prodtwo.setName("MAC Mini MGEQ2HN/A");
        prodtwo.setImage("http://i.ebayimg.com/00/s/NDEyWDU1MA==/z/6t8AA0Swq7JUAxCk/$_1");
        prodtwo.setDescription("It's mini in a massive way. Mac mini is an affordable pc");
        prodtwo.setCategory("Electronics");
        prodtwo.setPrice(69900);
        prodtwo.setQuantity(10);
        prodtwo.setShipping(500);
        prodtwo.setLocation("bangalore");

        products.add(prodone);
        products.add(prodtwo);
        return products;
    }
}
```



Step 7

- Generate/Refactor to extract interface Product Repository



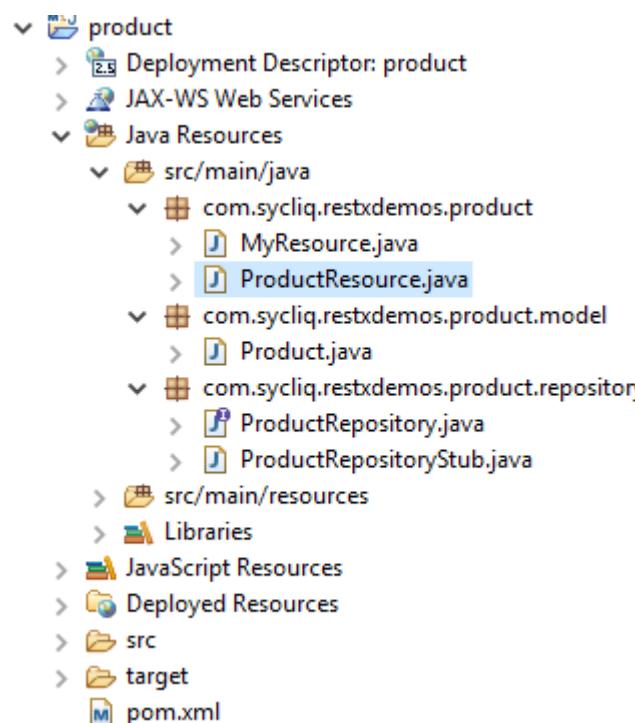
A screenshot of an IDE showing the 'ProductRepository.java' file. The code defines a public interface 'ProductRepository' with a single method 'findAllProducts()' that returns a list of 'Product'. The code is numbered from 1 to 11. The file tab for 'ProductRepository.java' is highlighted.

```
1 package com.sycliq.restdemos.product.repository;
2
3+ import java.util.List;
6
7 public interface ProductRepository {
8
9     List<Product> findAllProducts();
10
11 }
```



Step 8

- Create ProductResource service to render endpoint view



```
1 package com.sycliq.restxdemos.product;
2
3 import java.util.List;
4
5 import javax.ws.rs.GET;
6 import javax.ws.rs.Path;
7 import javax.ws.rs.Produces;
8 import javax.ws.rs.core.MediaType;
9
10 import com.sycliq.restxdemos.product.model.Product;
11 import com.sycliq.restxdemos.product.repository.ProductRepository;
12 import com.sycliq.restxdemos.product.repository.ProductRepositoryStub;
13
14 @Path("products")
15 public class ProductResource {
16
17     private ProductRepository productRepo = new ProductRepositoryStub();
18     @GET
19     @Produces(MediaType.APPLICATION_JSON)
20     public List<Product> getAllProducts(){
21         return productRepo.findAllProducts();
22     }
23
24
25 }
```



Step 9

- To Render JSON
 - MessageBodyWriter not found for media type=application/json
 - JSON requires another Jar to be added to our **pom.xml**
 - Jersey-media-moxy
 - **@Produces(MediaType.APPLICATION_JSON)**
 - Jersey uses JAXB to convert the POJO to XML and then XML to JSON
 - Our XML annotations affect JSON output



Step 10

- Edit the pom.xml to uncomment
- Org.glassfish.jersey.media
- Jersey-media-moxy

The screenshot shows an IDE interface with several tabs at the top: Product.java, ProductReposit..., ProductReposit..., ProductResourc..., Apache Tomcat/..., and product/pom.xml. The product/pom.xml tab is active, displaying the Maven configuration file. A red curly brace on the right side of the code highlights the Jersey-media-moxy dependency section, with the text "Uncomment this!" written next to it. The code itself is as follows:

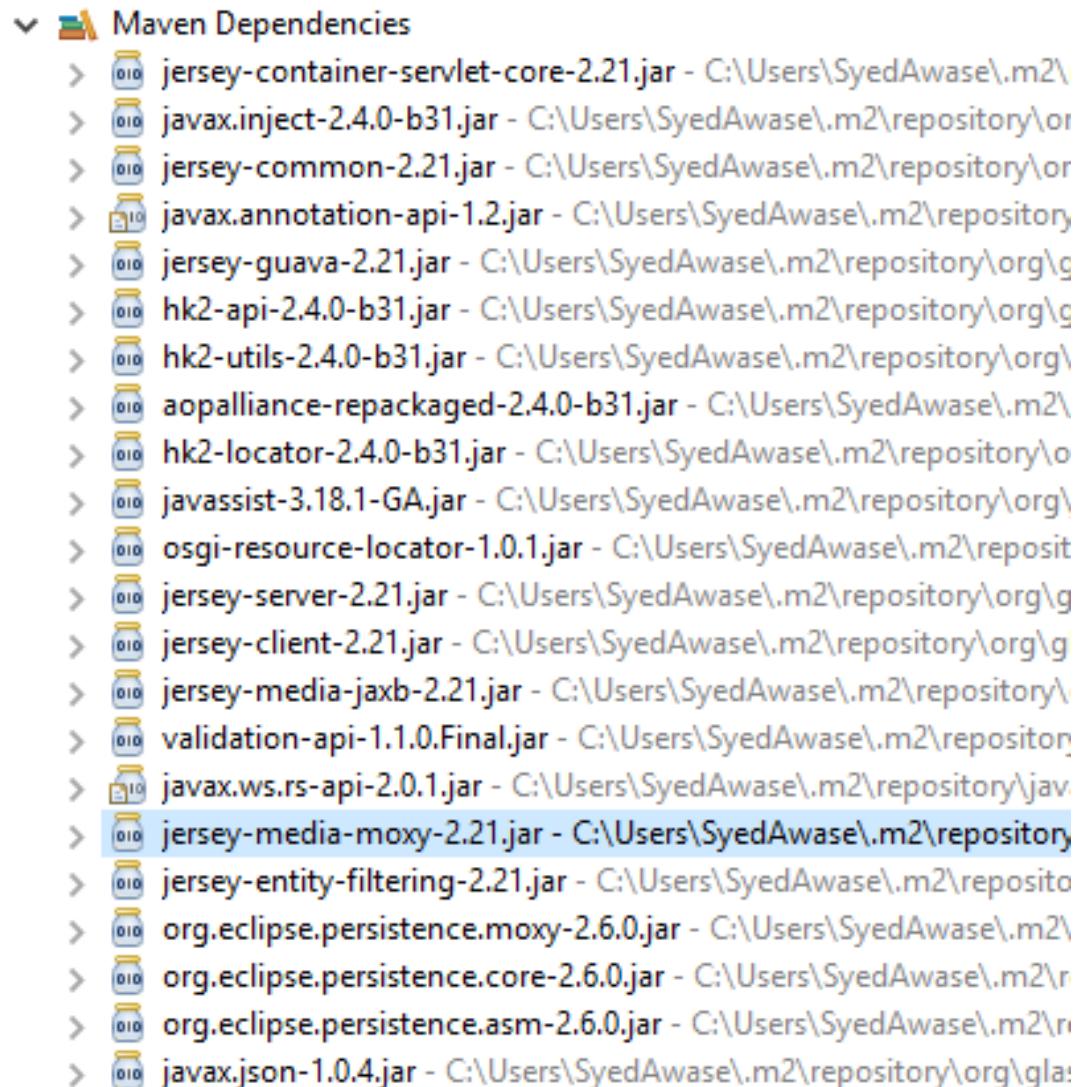
```
14      <plugins>
15          <plugin>
16              <groupId>org.apache.maven.plugins</groupId>
17              <artifactId>maven-compiler-plugin</artifactId>
18              <version>2.5.1</version>
19              <inherited>true</inherited>
20              <configuration>
21                  <source>1.7</source>
22                  <target>1.7</target>
23              </configuration>
24          </plugin>
25      </plugins>
26  </build>
27
28  <dependencyManagement>
29      <dependencies>
30          <dependency>
31              <groupId>org.glassfish.jersey</groupId>
32              <artifactId>jersey-bom</artifactId>
33              <version>${jersey.version}</version>
34              <type>pom</type>
35              <scope>import</scope>
36          </dependency>
37      </dependencies>
38  </dependencyManagement>
39
40  <dependencies>
41      <dependency>
42          <groupId>org.glassfish.jersey.containers</groupId>
43          <artifactId>jersey-container-servlet-core</artifactId>
44          <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
45          <!-- artifactId>jersey-container-servlet</artifactId -->
46      </dependency>
47      <!-- uncomment this to get JSON support
48      <dependency>
49          <groupId>org.glassfish.jersey.media</groupId>
50          <artifactId>jersey-media-moxy</artifactId>
51      </dependency>
52      -->
53  </dependencies>
54  <properties>
55      <jersey.version>2.21</jersey.version>
56      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
57  </properties>
58 </project>
```

Uncomment this!



Step 11

- Check for jersey-media-moxy package updated in maven repository.



Step 12

- Annotate Product Model with @XmlRootElement

```
package com.sycliq.restxdemos.product.model;

import javax.xml.bind.annotation.XmlRootElement;

/**
 * @author SyedAwase
 *
 */
@XmlRootElement
public class Product {

    private String name;
    private String image;
    private String description;
    private String category;
    private int price;
    private int quantity;
    private int shipping;
    private String location;
    public String getName() {
        return name;
    }
    public void setName(String name) {
```





Step 13

- run your application to render the endpoint view
- <http://localhost:8080/product/webapi/products/>

The screenshot shows the Postman application interface. At the top, there are tabs for 'NEW', 'Runner', 'Import', 'Builder' (which is selected), and 'Team Library'. Below the tabs, there's a toolbar with icons for 'IN SYNC', 'No Environment', and other settings. The main area shows a sequence of API requests: 'Get all brands', 'Get device spec', 'Get devices fro...', and 'Get devices fro...'. A 'GET' request is selected with the URL 'http://localhost:8080/product/webapi/products'. The 'Authorization' tab is active, showing 'No Auth'. The 'Body' tab is selected, displaying a JSON response:

```
1 [ { "category": "Electronics", "description": "With the OnePlus 2, we have something bold to say. We believe that great products come from great ideas, not multi-", "image": "http://i.ebayimg.com/00/s/Mzc4lDcyMA==/z/aUoAAOSwyQtV4-Kg$_.12.JPG", "location": "Punjab", "name": "OnePlus Two 64GB", "price": 34999, "quantity": 1, "shipping": 100 }, { "category": "Electronics", "description": "It's mini in a massive way. Mac mini is an affordable powerhouse that packs the entire Mac experience into a 19.7cm", "image": "http://i.ebayimg.com/00/s/NDEyWDU1M4==/z/6t8AAOSwq7JUAxCK$_.12.JPG", "location": "bangalore", "name": "MAC Mini MGEOQ2HN/A", "price": 69900, "quantity": 10, "shipping": 500 } ]
```

The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 55 ms'.



@Produces

- @Produces annotation can be used to produce multiple output formats.
- @Produces can take a String or an Array of Strings
- @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
- REST Clients
 - Postman (Chrome)
 - Poster (Firefox)
 - Fiddler (Telerik)

JERSEY

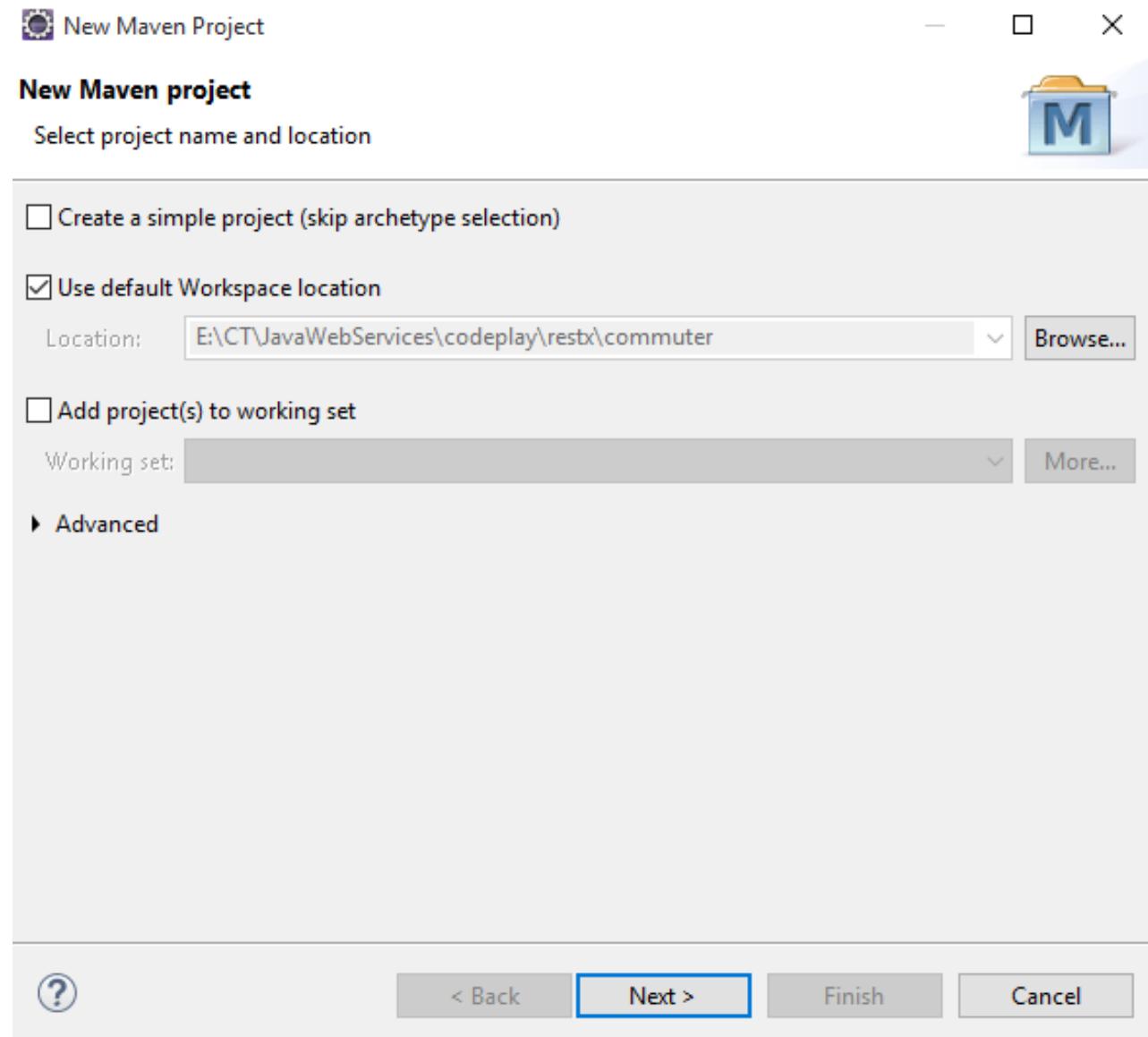
IX(D): PLAY BOOK: GET BY ID –BOOK SERVICE

```
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})  
@PathParam("bookId")
```



Step 1:

- create a Maven Project



Step 2:

- Maven Project of Archetype
 - Org.glassfish.jersey.archetypes
 - Jersey-quickstart-webapp
 - 2.21 version

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

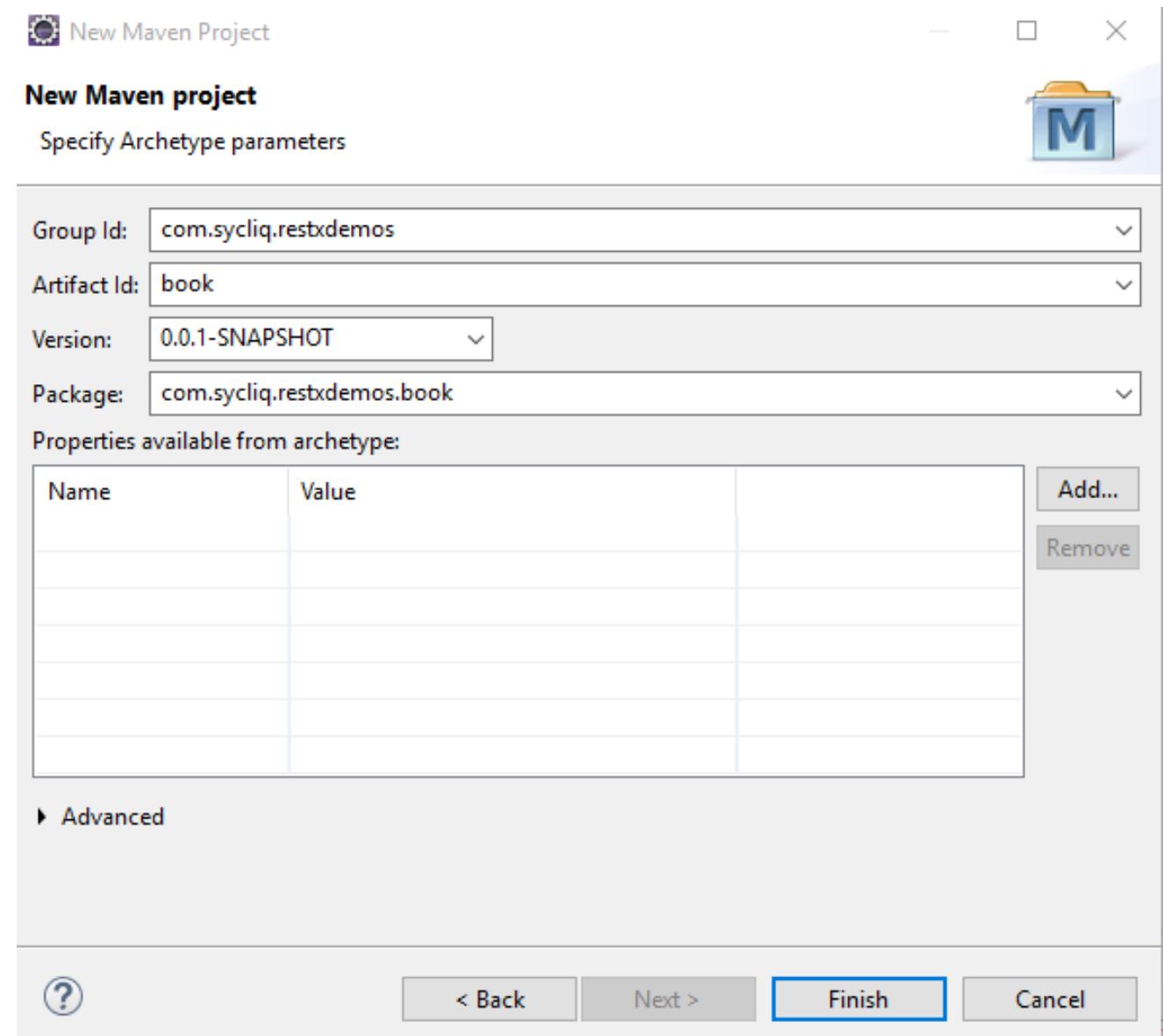
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0
org.glassfish.jersey.archetypes	jersey-quickstart-webapp	2.21

Show the last version of Archetype only Include snapshot archetypes

► Advanced

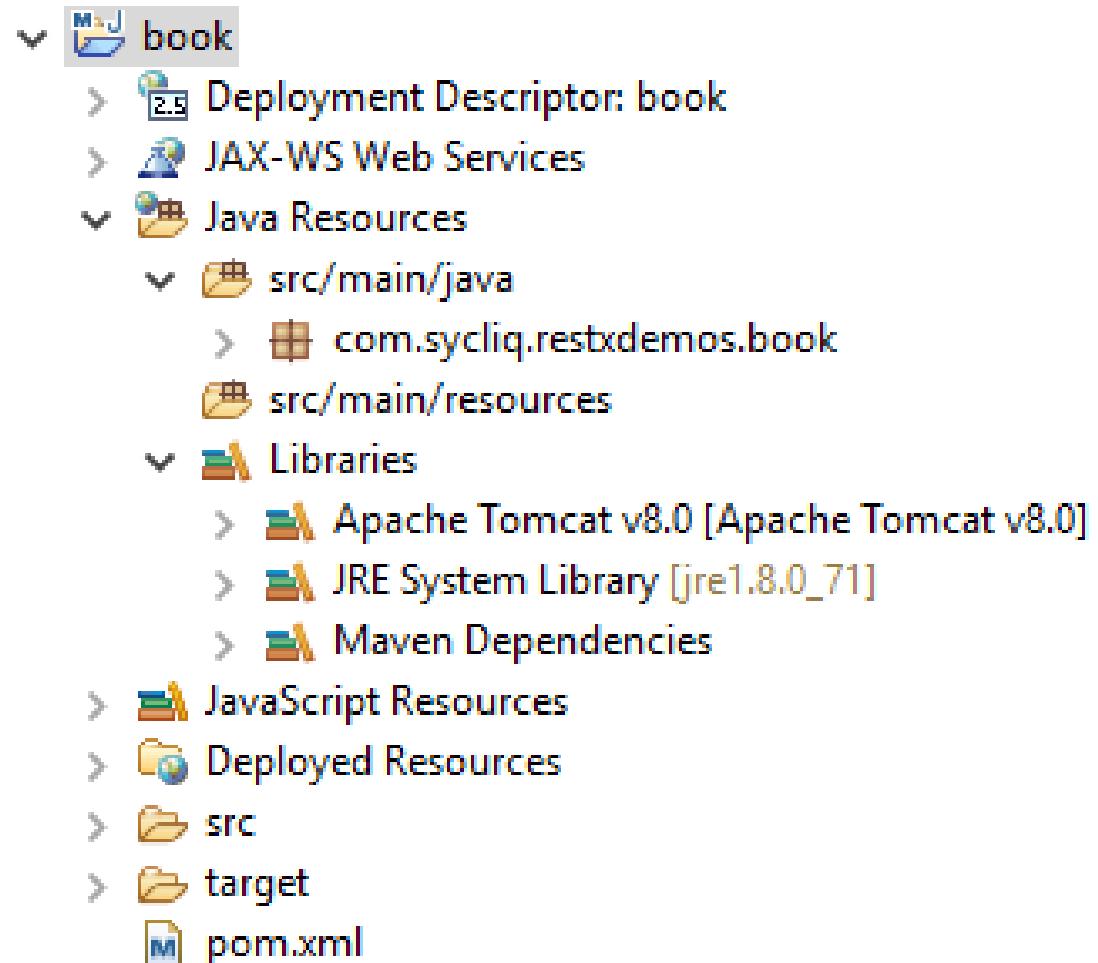


Step 3:



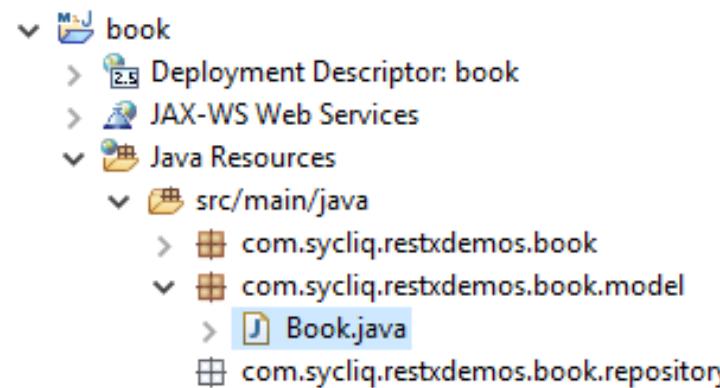
Step 4

- scaffolds a maven archetype project for jersey-quickstart-webapp



Step 5

- create Book model



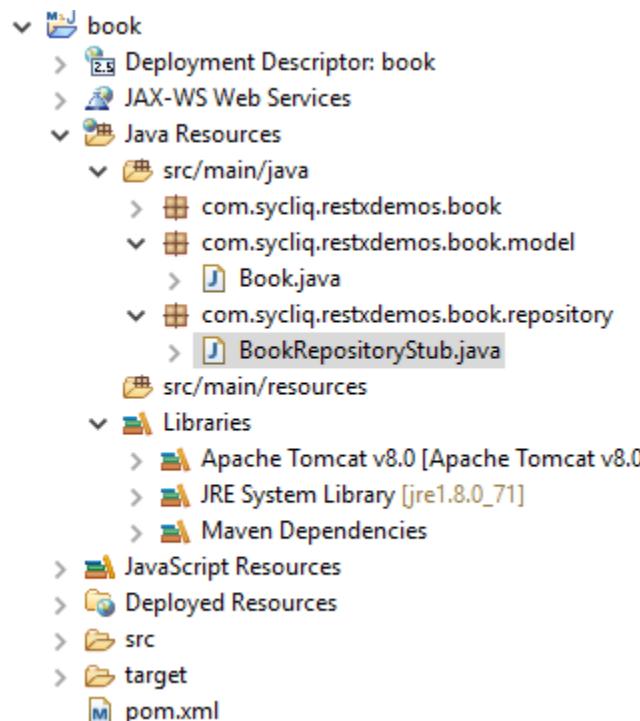
```
+ /**
+  * @author SyedAwase
+  */
+
@XmlRootElement
public class Book {

    private int bookId;
    private String bookTitle;
    private String bookAuthor;
    private String bookPublisher;
    private String bookEditor;
    private int bookPrice;
    public int getBookId() {
        return bookId;
    }
    public void setBookId(int bookId) {
        this.bookId = bookId;
    }
    public String getBookTitle() {
        return bookTitle;
    }
    public void setBookTitle(String bookTitle) {
        this.bookTitle = bookTitle;
    }
    public String getBookAuthor() {
        return bookAuthor;
    }
    public void setBookAuthor(String bookAuthor) {
        this.bookAuthor = bookAuthor;
    }
}
```



Step 6

- create
BookRepositoryStub



```
import com.syqliq.restxdemos.book.model.Book;

/**
 * @author SyedAwase
 */
public class BookRepositoryStub {

    public List<Book> findAllBooks(){

        List<Book> books = new ArrayList<Book>();

        Book bookOne = new Book();
        bookOne.setBookId(1);
        bookOne.setBookTitle("Fountain Head");
        bookOne.setBookAuthor("Ayn Rand");
        bookOne.setBookEditor("XYZ");
        bookOne.setBookPublisher("McGrawHill");
        bookOne.setBookPrice(250);

        Book bookTwo = new Book();
        bookTwo.setBookId(2);
        bookTwo.setBookTitle("Systems for Crop Life Cycle Intelligence");
        bookTwo.setBookAuthor("Syed Awase");
        bookTwo.setBookEditor("MNQ");
        bookTwo.setBookPublisher("McGrawHill");
        bookTwo.setBookPrice(850);

        books.add(bookOne);
        books.add(bookTwo);

        return books;
    }

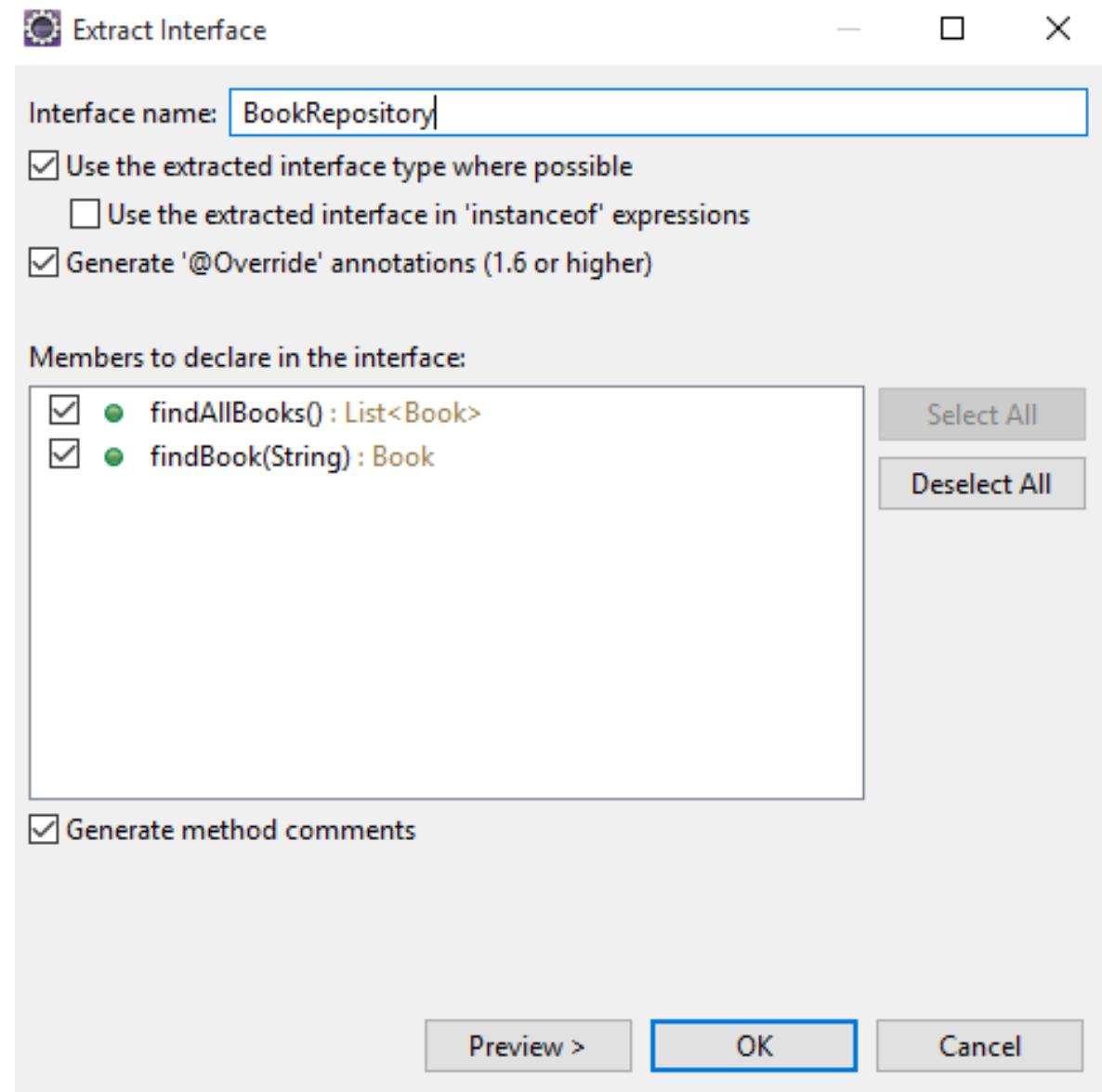
    public Book findBook(String bookId) {
        Book bookTwo = new Book();
        bookTwo.setBookId(2);
        bookTwo.setBookTitle("Systems for Crop Life Cycle Intelligence");
        bookTwo.setBookAuthor("Syed Awase");
        bookTwo.setBookEditor("MNQ");
        bookTwo.setBookPublisher("McGrawHill");
        bookTwo.setBookPrice(850);

        return bookTwo;
    }
}
```



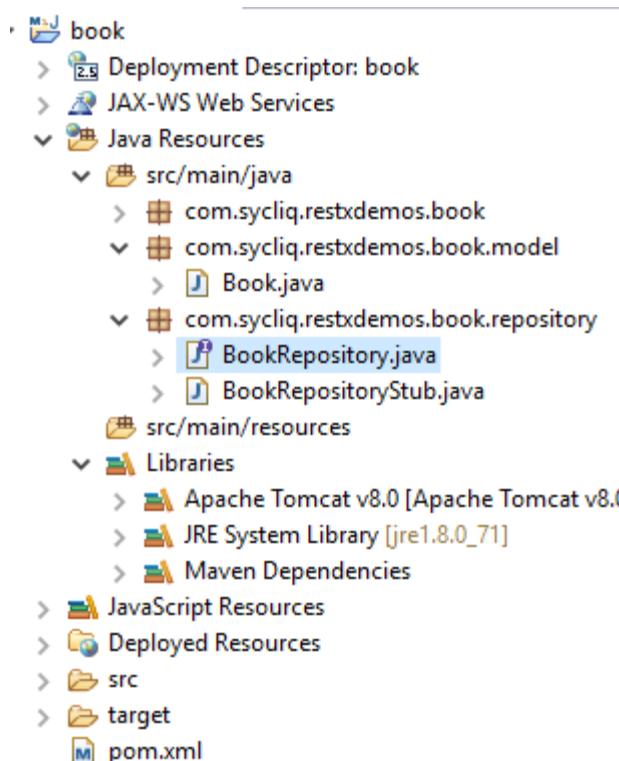
Step 7

- Extract BookRepository Interface from BookRepositoryStub



Step 8

- extract BookRepository Interface

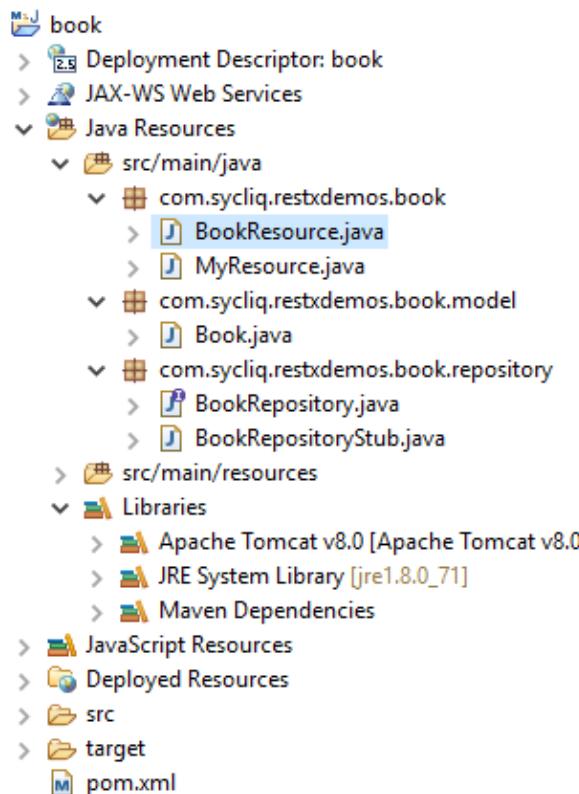


```
1 package com.sycliq.restdemos.book.repository;  
2  
3+ import java.util.List;  
4  
7 public interface BookRepository {  
8  
9     List<Book> findAllBooks();  
10  
11    Book findBook(String bookId);  
12  
13 }
```



Step 9:

- create BookResource endpoint view to render a web service

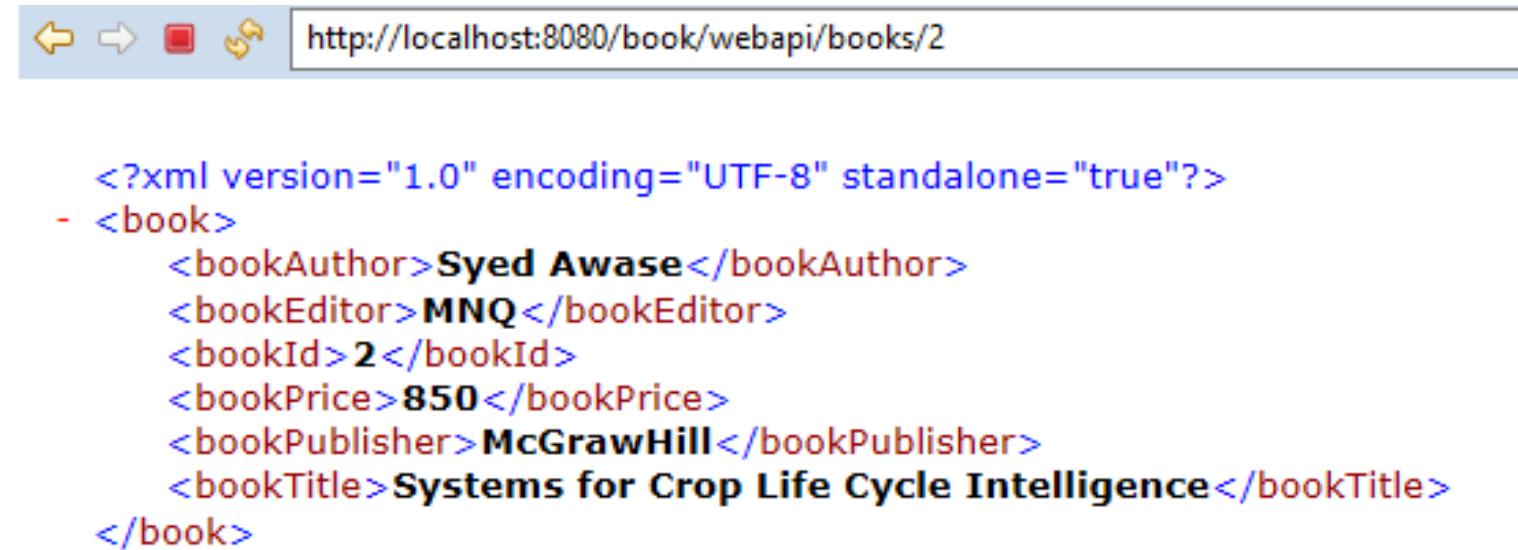


```
1+ /**
2 package com.sycliq.restxdemos.book;
3
4 import java.util.List;
5
6 import javax.websocket.serverPathParam;
7 import javax.ws.rs.GET;
8 import javax.ws.rs.Path;
9 import javax.ws.rs.Produces;
10 import javax.ws.rs.core.MediaType;
11
12 import com.sycliq.restxdemos.book.model.Book;
13 import com.sycliq.restxdemos.book.repository.BookRepository;
14 import com.sycliq.restxdemos.book.repository.BookRepositoryStub;
15
16 /**
17 * @author SyedAwase
18 */
19
20
21
22 @Path("books")
23 public class BookResource {
24
25     private BookRepository bookRepository = new BookRepositoryStub();
26
27     @GET
28     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
29     public List<Book> getAllBooks() {
30         return bookRepository.findAllBooks();
31     }
32
33     @GET
34     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
35     @PathParam("{bookId}")
36     public Book getBook(@PathParam("bookId") String bookId) {
37         return bookRepository.findBook(bookId);
38     }
39 }
```



Step 10

- run the server to render the end point service
- <http://localhost:8080/book/webapi/books/2>



The screenshot shows a web browser window with the URL `http://localhost:8080/book/webapi/books/2` in the address bar. The page content displays an XML document representing a book. The XML is color-coded: blue for tags like `<?xml`, `<book>`, and `<bookAuthor>`; red for values like `version="1.0"`, `encoding="UTF-8"`, `standalone="true"`, `Syed Awase`, `MNQ`, `2`, `850`, `McGrawHill`, `Systems for Crop Life Cycle Intelligence`; and black for other tags like `</book>`.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <book>
  <bookAuthor>Syed Awase</bookAuthor>
  <bookEditor>MNQ</bookEditor>
  <bookId>2</bookId>
  <bookPrice>850</bookPrice>
  <bookPublisher>McGrawHill</bookPublisher>
  <bookTitle>Systems for Crop Life Cycle Intelligence</bookTitle>
</book>
```



@Consumes

- This annotation is used to specify which MIME media types of representations a resource can accept or consume from the client.
- If @Consumes is applied at the class level, all the response methods accept the specified MIME types by default
- If @Consumes is applied at the method level, it overrides any @Consumes annotations applied at the class level.
- Determining what a method will consume
 - Application/x-www-form-urlencoded
 - Application/json
 - Application/xml
 - @Consumes(MediaType.APPLICATION_FORM_URLENCODED)



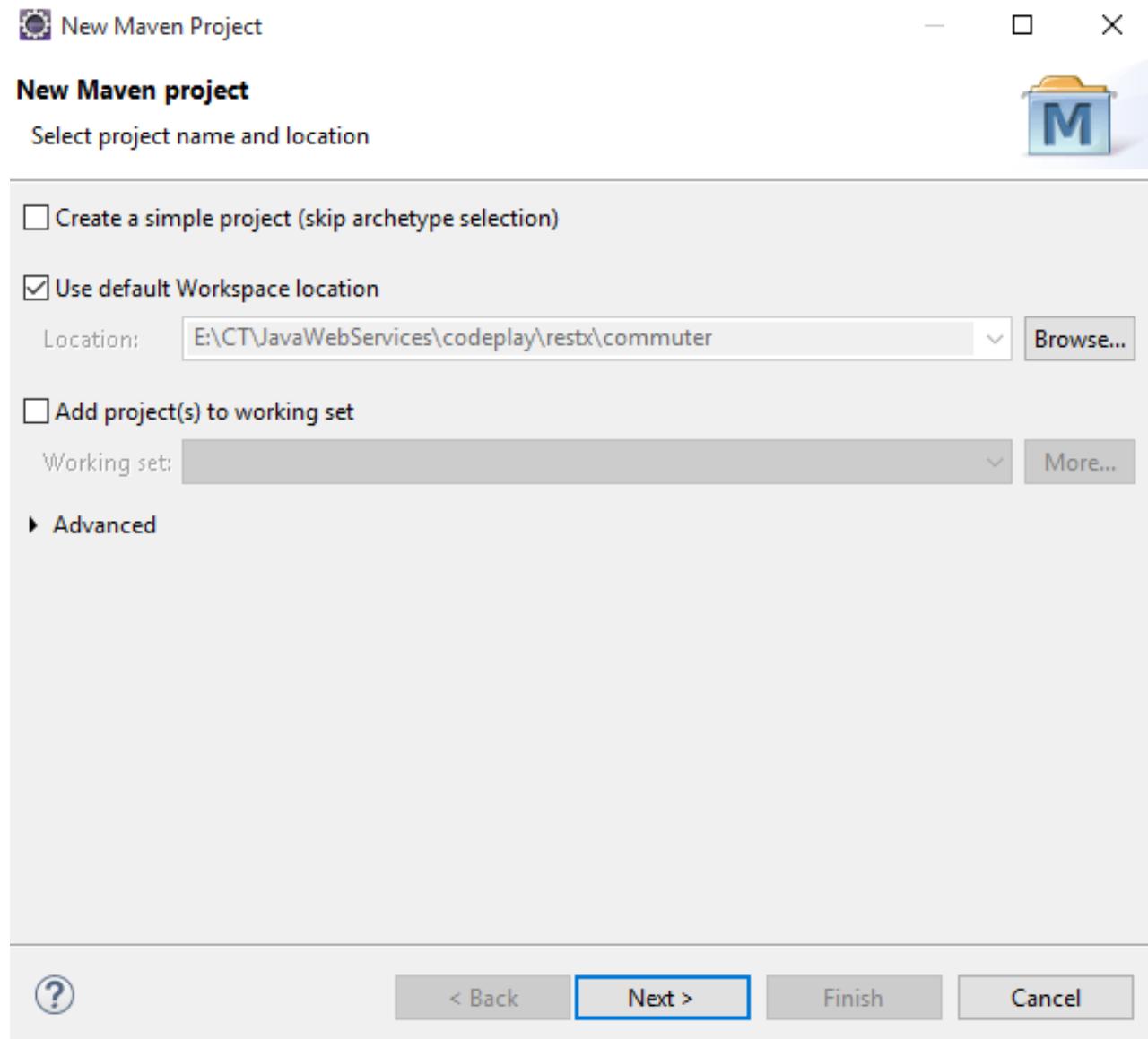
JERSEY

IX(E): PLAY BOOK: POST + @CONSUMES DEMO : CUSTOMER



Step 1:

- create a Maven Project



Step 2:

- Maven Project of Archetype
 - Org.glassfish.jersey.archetypes
 - Jersey-quickstart-webapp
 - 2.21 version

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

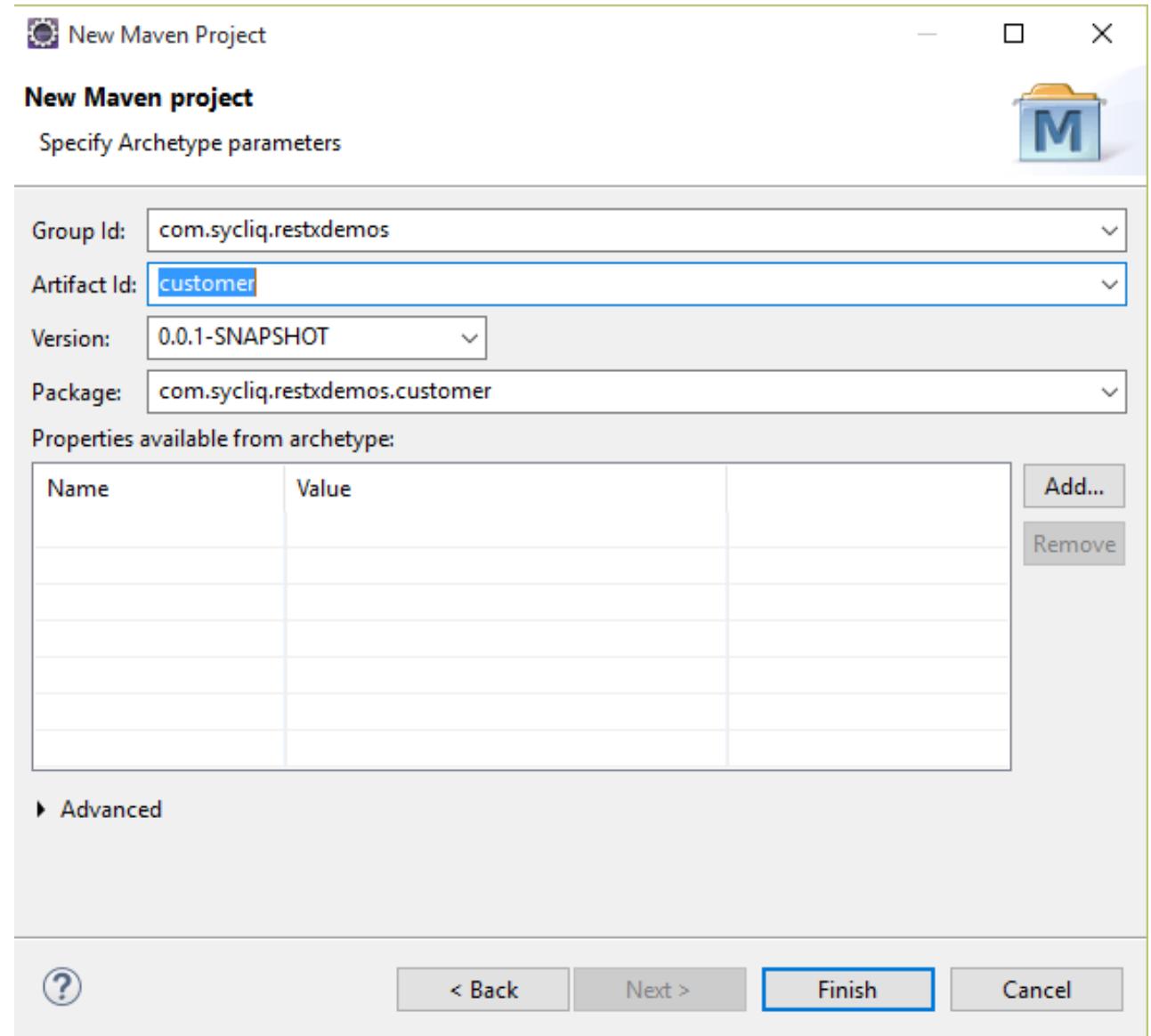
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0
org.glassfish.jersey.archetypes	jersey-quickstart-webapp	2.21

Show the last version of Archetype only Include snapshot archetypes

► Advanced

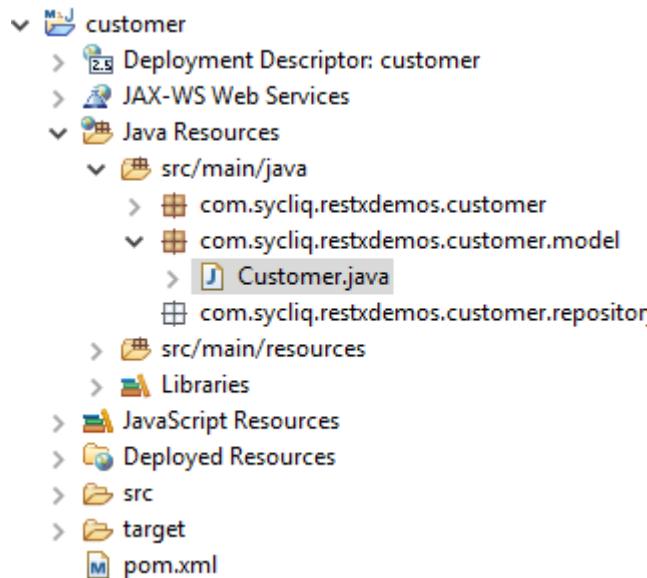


Step 3



Step 4

- Create Customer Model



```
package com.syqliq.restxdemos.customer.model;

import javax.xml.bind.annotation.XmlRootElement;

/** * @author SyedAwase | * */
@XmlRootElement
public class Customer {

    private int customerId;
    private String customerName;
    private String customerEmail;
    private String customerCountry;

    public int getCustomerId() {
        return customerId;
    }

    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }

    public String getCustomerEmail() {
        return customerEmail;
    }

    public void setCustomerEmail(String customerEmail) {
        this.customerEmail = customerEmail;
    }

    public String getCustomerCountry() {
        return customerCountry;
    }

    public void setCustomerCountry(String customerCountry) {
        this.customerCountry = customerCountry;
    }
}
```



Step

The screenshot shows the Postman application interface in 'Builder' mode. The top navigation bar includes 'NEW', 'Runner', 'Import', 'Builder' (which is highlighted), 'Team Library', and various status indicators like 'IN SYNC'. The main area displays a POST request to 'http://localhost:8080/customer/webapi/customers/createcustomer'. The 'Body' tab is selected, showing four form-data fields: 'customerId' (value: 812109132), 'customerName' (value: Syed Ameese Sadath), 'customerEmail' (value: sas@sycliq.com), and 'customerCountry' (value: USA). Below the body, the 'Test Results' section shows a successful response with status 200 OK and time 58 ms. The JSON response is displayed in a pretty-printed format:

```
1 {  
2   "customerCountry": "USA",  
3   "customerEmail": "sas@sycliq.com",  
4   "customerId": "812109132",  
5   "customerName": "Syed Ameese Sadath"  
6 }
```



JERSEY

IX(F): PLAY BOOK: CRUD OPERATIONS WITH JERSEY

PUT

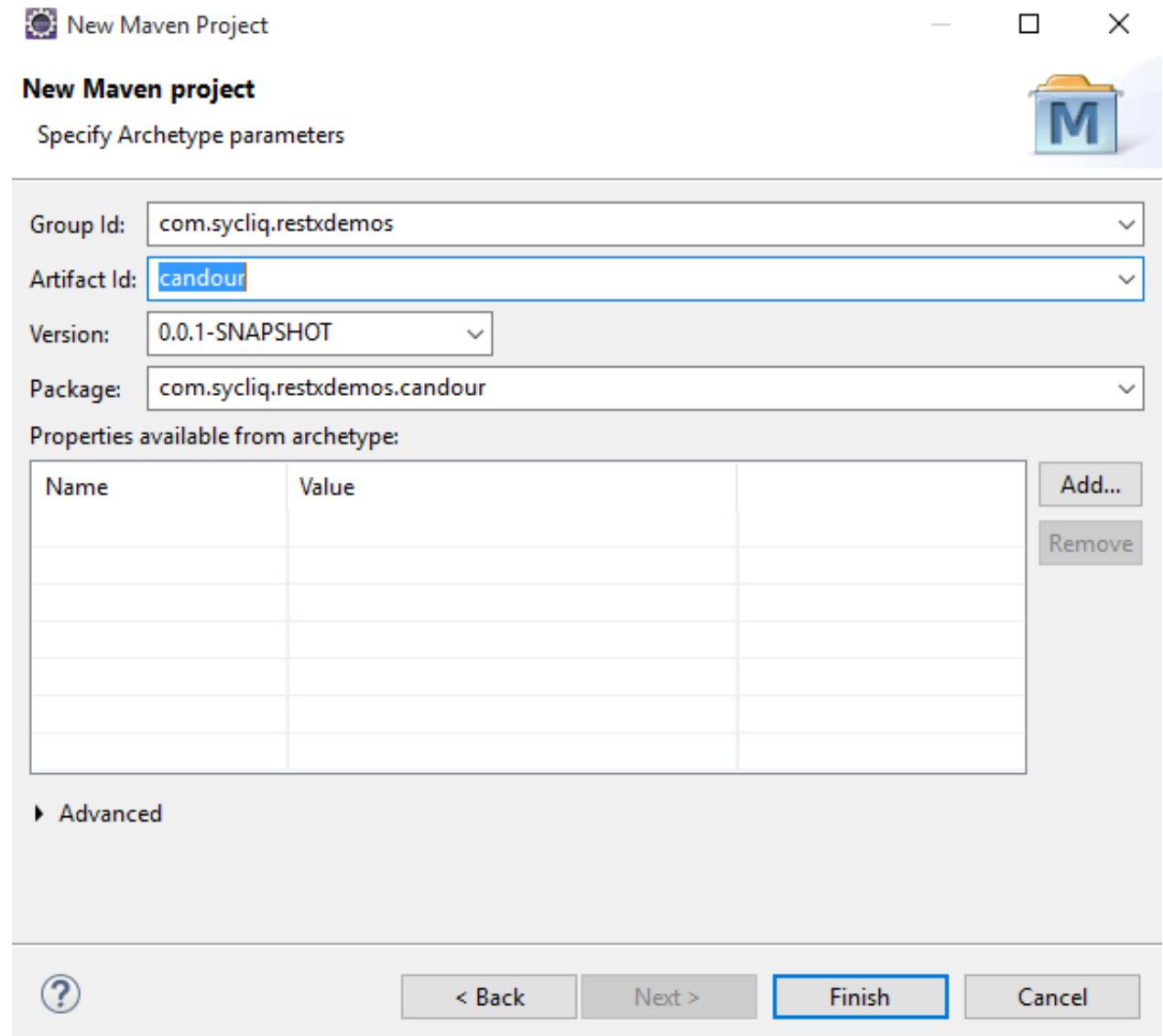
- Some browsers do not support PUT(or DELETE)
- Often we use POST to CREATE and UPDATE records
- PUT is **idempotent**
 - We can call PUT multiple times and it should not affect the app.
 - Similar to SQL Update
 - PUT should ONLY be used if you are supplying the ID for your Object

- PUT Should not be used if objectId is sequential being generated.
 - POST should be used if the SERVER is supplying the objectId.
 - POST to a generic URL, where each call will create a new Object
 - PUT to a specific URL, where a call will either create or update an Object (Idempotent)
- **GET /PUT/DELETE(SAFELY repeatable)**
- **POST(non-repeatable)**



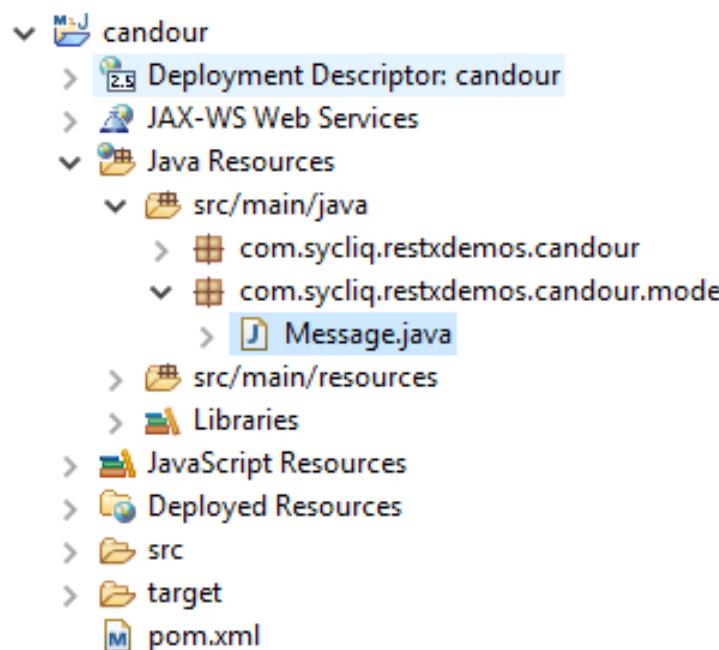
Step 1

- create a maven project for jersey



Step 2

- create models



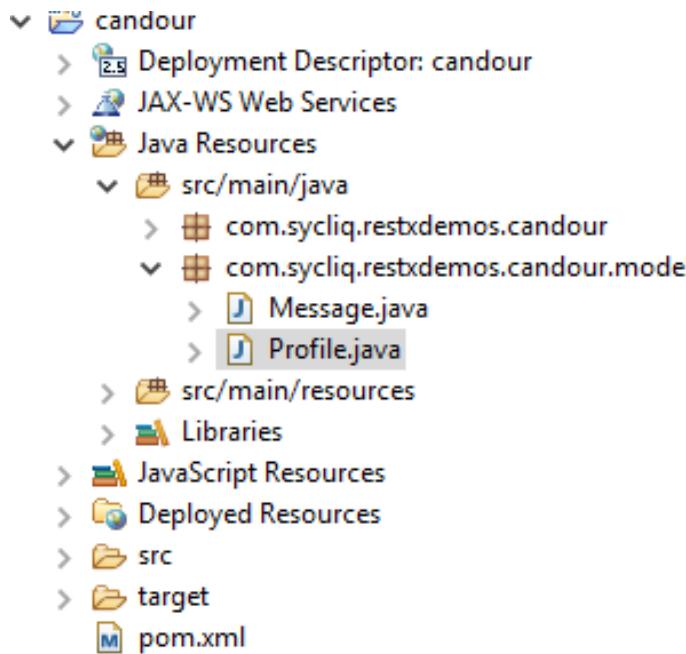
```
package com.syqliq.restxdemos.candour.model;
import java.util.Date;
public class Message {
    private long id;
    private String message;
    private Date created;
    private String author;

    public Message() {
    }
    public Message(long id, String message, Date created, String author) {
        this.id = id;
        this.message = message;
        this.author = author;
        this.created = new Date();
    }
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public Date getCreated() {
        return created;
    }
    public void setCreated(Date created) {
        this.created = created;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
}
```



Step 3

- create model for Profile



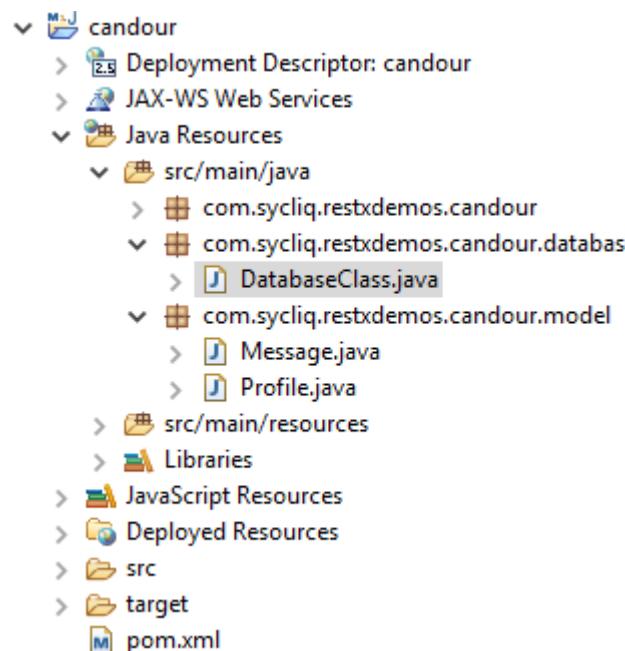
```
package com.sycliq.restxdemos.candour.model;
import java.util.Date;
public class Profile {

    private long id;
    private String profileName;
    private String firstName;
    private String lastName;
    private Date created;
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getProfileName() {
        return profileName;
    }
    public void setProfileName(String profileName) {
        this.profileName = profileName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public Date getCreated() {
        return created;
    }
    public void setCreated(Date created) {
        this.created = created;
    }
    public Profile() { }
    public Profile(long id, String profileName, String firstName, String lastName, Date created) {
        this.id=id;
        this.profileName=profileName;
        this.firstName=firstName;
        this.lastName=lastName;
    }
}
```



Step 4

- create database class for in memory data storage

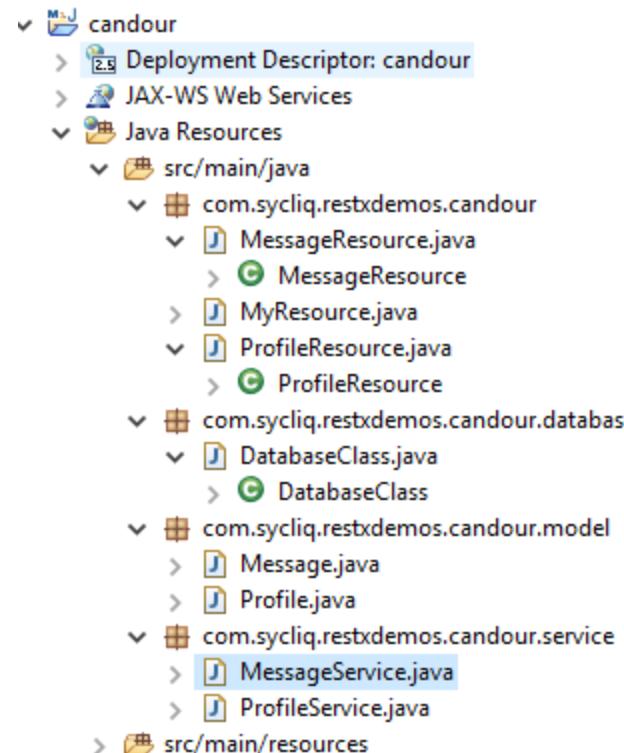


```
package com.sycliq.restxdemos.candour.database;  
+ import java.util.HashMap;  
  
public class DatabaseClass {  
  
    private static Map<Long,Message> messages = new HashMap<>();  
    private static Map<String,Profile> profiles = new HashMap<>();  
  
    public static Map<Long, Message> getMessages(){  
        return messages;  
    }  
  
    public static Map<String,Profile> getProfiles(){  
        return profiles;  
    }  
}
```



Step 5

- create Services for Message and Profile
- MessageService



```
package com.sycliq.restxdemos.candour.service;

import java.util.ArrayList;

public class MessageService {
    private Map<Long, Message> messages = DatabaseClass.getMessages();

    public MessageService() {
        messages.put(1L,new Message((long) 1, "SycliQ Launched",new Date(), "Syed Awase"));
        messages.put(2L,new Message((long) 2, "Aspire Launched",new Date(), "Syed Ameese"));
        messages.put(3L,new Message((long) 3, "TruEstate Launched",new Date(), "Syed Azeez"));
    }

    public List<Message> getAllMessages(){
        return new ArrayList<Message>(messages.values());
    }

    public Message getMessage(long messageId) {
        return messages.get(messageId);
    }

    public Message addMessage(Message message) {
        message.setId((long) (messages.size()+1));
        messages.put(message.getId(), message);
        return message;
    }

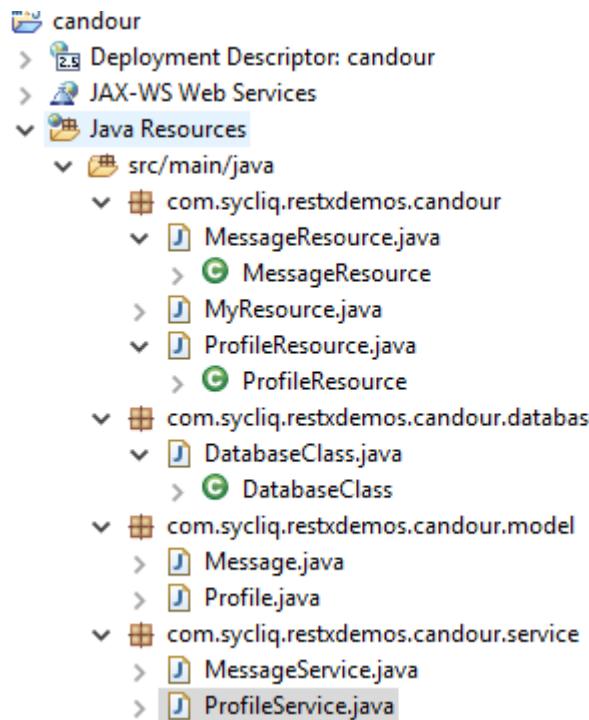
    public Message updateMessage(Message message) {
        if(message.getId() <= 0) {
            return null;
        }
        messages.put(message.getId(),message);
        return message;
    }

    public Message removeMessage(long id) {
        return messages.remove(id);
    }
}
```



Step 6

- Create Profile Service



```
package com.syqliq.restdemos.candour.service;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;

import com.syqliq.restdemos.candour.database.DatabaseClass;
import com.syqliq.restdemos.candour.model.Profile;

public class ProfileService {

    private Map<String, Profile> profiles =DatabaseClass.getProfiles();

    public ProfileService() {
        profiles.put("Syed", new Profile(1L,"Syed Awase", "Syed Awase", "Awase", new Date()));
    }

    public List<Profile> getAllProfiles(){
        return new ArrayList<Profile>(profiles.values());
    }

    public Profile getProfile(String profileName) {
        return profiles.get(profileName);
    }

    public Profile addProfile(Profile profile) {
        profile.setId((long) (profiles.size()+1));
        profiles.put(profile.getProfileName(), profile);
        return profile;
    }

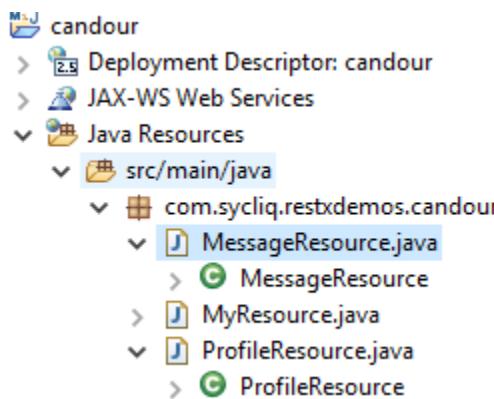
    public Profile updateProfile(Profile profile) {
        if(profile.getProfileName().isEmpty()) {
            return null;
        }
        profiles.put(profile.getProfileName(), profile);
        return profile;
    }

    public Profile removeProfile(String profileName) {
        return profiles.remove(profileName);
    }
}
```



Step 7

- create MessageResource to map to CRUD operations



```
package com.sycliq.restxdemos.candour;

import java.util.List;

@Path("/messages")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class MessageResource {

    MessageService messageService = new MessageService();

    @GET
    public List<Message> getMessages(){
        return messageService.getAllMessages();
    }

    @GET
    @Path("/{ messageId }")
    public Message getMessage(@PathParam("messageId") long id) {
        return messageService.getMessage(id);
    }

    @POST
    public Message addMessage(Message message) {
        return messageService.addMessage(message);
    }

    @PUT
    @Path("/{ messageId }")
    public Message updateMessage(@PathParam("messageId") long id,Message message) {
        message.setId(id);
        return messageService.updateMessage(message);
    }

    @DELETE
    @Path("/{ messageId }")
    public String deleteMessage(@PathParam("messageId") long id) {
        messageService.removeMessage(id);
        String successMxg="Successfully deleted a record of::"+id;
        return successMxg;
    }
}
```



Step 8:

- create ProfileService

```
@Path("/profiles")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class ProfileResource {

    private ProfileService profileService = new ProfileService();

    @GET
    public List<Profile> getProfiles(){
        return profileService.getAllProfiles();
    }

    @POST
    public Profile addProfile(Profile profile) {
        return profileService.addProfile(profile);
    }

    @GET
    @Path("/{profileName}")
    public Profile getProfile(@PathParam("profileName") String profileName) {
        return profileService.getProfile(profileName);
    }

    @PUT
    @Path("/{profileName}")
    public Profile updateProfile(@PathParam("profileName") String profileName, Profile profile) {
        profile.setProfileName(profileName);
        return profileService.updateProfile(profile);
    }

    @DELETE
    @Path("/{profileName}")
    public void deleteProfile(@PathParam("profileName") String profileName) {
        profileService.removeProfile(profileName);
    }
}
```



Step 9

- uncomment jersey-media-moxy to render JSON MIME Type
- Subsequently Annotate models Message and Profile with XmlRootElement

```
<!-- uncomment this to get JSON support-->
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-moxy</artifactId>
</dependency>

import java.util.Date;
@XmlRootElement
public class Message {
    private Long id;
    private String message;
    private Date created;
    private String author;

    import java.util.Date;
    @XmlElement
    public class Profile {

        private Long id;
        private String profileName;
        private String firstName;
        private String lastName;
        private Date created;
```



Step 10

- run your application to render endpoint urls
 - <http://localhost:8080/candour/webapi/messages>
 - <http://localhost:8080/candour/webapi/profiles>

The screenshot shows a browser window with the URL `http://localhost:8080/candour/webapi/profiles`. The response is a JSON array containing three profile objects:

```
[{"id": 1, "firstName": "Syed Awase", "lastName": "Awase", "profileName": "Syed Awase"}, {"id": 2, "firstName": "Admin for j2ee", "lastName": "User", "profileName": "admin"}, {"id": 3, "firstName": "developer", "lastName": "User dev", "profileName": "developer"}]
```



Filtering by Year

- Add a method to `getAllMessagesforYear` in `MessageService`

```
//to fetch all messages by year
public List<Message> getAllMessagesforYear(int year){  
  
    List<Message> messagesForYear = new ArrayList<>();  
    Calendar cal = Calendar.getInstance();  
    for(Message message:messages.values()) {  
        cal.setTime(message.getCreated());  
        if(cal.get(Calendar.YEAR)==year) {  
            messagesForYear.add(message);  
        }  
    }  
    return messagesForYear;  
}
```



Pagination

- retrieve messages based on starting and no of messages to display for MessageService

```
public List<Message> getAllMessagesPaginated(int start, int size){  
    ArrayList<Message> list = new ArrayList<Message>(messages.values());  
    if(start+size> list.size()) return new ArrayList<Message>();  
    return list.subList(start,start +size);  
}
```



MessageResource

- **Modify MessageResource**

to render based on

QueryParameters/Filtering

@QueryParams

```
@GET  
public List<Message> getMessages(@QueryParam("year") int year){  
    if(year>0) {  
        return messageService.getAllMessagesforYear(year);  
    }  
    return messageService.getAllMessages();  
}
```



MessageResource

- **QueryParameters** for rendering based on pagination.

@QueryParams

```
@GET  
public List<Message> getMessages(@QueryParam("year") int year,  
                                 @QueryParam("start") int start, @QueryParam("size") int size){  
    if(year>0) {  
        return messageService.getAllMessagesforYear(year);  
    }  
    if(start>=0 && size>=0) {  
        return messageService.getAllMessagesPaginated(start, size);  
    }  
    return messageService.getAllMessages();  
}
```



@MatrixParams

```
@GET  
@Path("annotations")  
@Produces(MediaType.TEXT_PLAIN)  
public String getParamsUsingAnnotations(@MatrixParam("param") String matrixParam) {  
    return ("Matrix Param:" + matrixParam);  
}
```

http://localhost:8080/candour/webapi/myresource/annotations;param="this is from matrix param"

The screenshot shows the Postman application interface. At the top, there's a header bar with the URL 'http://localhost:8080/' and a 'No Environment' dropdown. Below the header, a main panel has 'GET' selected in a dropdown, the URL 'http://localhost:8080/candour/webapi/myresource/annotations;param="this is from matrix param"', and a 'Send' button. Underneath, tabs for 'Authorization', 'Headers' (which is active), 'Body', 'Pre-request Script', and 'Tests' are visible. A table below the tabs shows a single row with a 'Key' column containing 'New key' and a 'Value' column containing 'Value'. At the bottom, tabs for 'Body' (which is active), 'Cookies', 'Headers (4)', and 'Test Results' are shown, along with a status message 'Status: 200 OK'. At the very bottom, there are buttons for 'Pretty', 'Raw', 'Preview', and 'Text' (with a dropdown arrow), and a small icon.



@HeaderParam

```
@GET  
@Path("sessioninfo")  
@Produces(MediaType.TEXT_PLAIN)  
public String getSessionParams(@HeaderParam("authSessionId") String sessionToken) {  
    return ("Header-Parameter-SessionInfo:" + sessionToken);  
}
```

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:8080/`. Below the URL bar, there are buttons for `+` and `...`, and a dropdown for `NO ENVIRONMENT`. The main area shows a `GET` request to `http://localhost:8080/candour/webapi/myresource/sessioninfo`. To the right of the URL, there are `Params` and `Send` buttons. Below the URL, the `Headers (1)` tab is selected, showing a table with one row:

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> authSessionId	123jkjkasdhklasd12123			
New key	Value	Description		

Below the headers, the `Body` tab is selected, showing the status `Status: 200 OK`. The response body contains the value `Header-Parameter-SessionInfo:123jkjkasdhklasd12123`.



@CookieParam

```
@GET  
@Path("cookieinfo")  
@Produces(MediaType.TEXT_PLAIN)  
public String getCookieParams(@CookieParam("cookieVal") String cookieData) {  
    return ("Cookie-Parameter-Data:"+ cookieData);  
}
```



@Context

```
@GET  
@Path("contextdemo")  
@Produces(MediaType.TEXT_PLAIN)  
public String getContextParams(@Context UriInfo uriinfo) {  
    String mycontextpath= uriinfo.getAbsolutePath().toString();  
    return mycontextpath;  
}
```

The screenshot shows the Postman application interface. At the top, there's a header bar with tabs for 'New Tab', 'No Environment', and settings. Below that, a main toolbar has 'GET', a URL input field ('http://localhost:8080/candour/webapi/myresource/contextdemo'), 'Params', 'Send', and 'Save' buttons. Underneath, there are tabs for 'Authorization', 'Headers' (which is currently selected), 'Body', 'Pre-request Script', and 'Tests'. The 'Headers' table has a single row with 'Content-Type' as the key and 'application/json' as the value. In the 'Body' section, the status is shown as 'Status: 200 OK' and 'Time: 24 ms'. At the bottom, there are buttons for 'Pretty', 'Raw', and 'Preview'.

<http://localhost:8080/candour/webapi/myresource/contextdemo>



@Context

```
@GET  
@Path("contextdemo")  
@Produces(MediaType.TEXT_PLAIN)  
public String getContextParams(@Context UriInfo uriinfo, @Context HttpHeaders headers) {  
    String mycontextpath= uriinfo.getAbsolutePath().toString();  
    String cookies= headers.getCookies().toString();  
    return "path::"+mycontextpath+":::::"+"Cookies:::"+cookies;  
}
```

The screenshot shows the Postman application interface. At the top, there's a header bar with tabs for 'New Tab', '+', '...', and 'No Environment'. Below that is a search bar with 'GET' dropdown and a URL field containing 'http://localhost:8080/candour/webapi/myresource/contextdemo'. To the right of the URL are 'Params' and a 'Send' button. Underneath, there are tabs for 'Authorization', 'Headers' (which is currently selected), 'Body', 'Pre-request Script', and 'Tests'. In the 'Headers' section, there's a table with columns 'Key', 'Value', and 'Description'. A single row is present with 'Key' as 'New key', 'Value' as 'Value', and 'Description' as 'Description'. At the bottom, there are tabs for 'Body' (selected), 'Cookies', 'Headers (4)', 'Test Results', and a status indicator 'Status: 200 OK'. Below the table, there are buttons for 'Pretty', 'Raw', and 'Preview'.

path::http://localhost:8080/candour/webapi/myresource/contextdemo::::Cookies:{}



JERSEY

IX(G): PLAY BOOK: CREATING A SUB-RESOURCE





Sub-Resource

- /messages/{messageId}/Comments/

MESSAGERESOURCE

```
//path for sub-resource
@GET
@Path("/{messageId}/comments")
public CommentResource getCommentResource() {
    return new CommentResource();
}
```

COMMENTRESOURCE

```
package com.sycliq.restxdemos.candour;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;

@Path("/")
public class CommentResource {

    @GET
    public String getAllComments() {
        return "get All Comments";
    }

    @GET
    @Path("/{commentId}")
    public String getCommentById(@PathParam("messageId") long messageId,@PathParam("commentId") long commentId) {
        return "get Comment by Id:"+commentId+"MessageId::"+messageId;
    }
}
```

Comment Model



```
package com.sycliq.restdemos.candour.model;

import java.util.Date;

public class Comment {

    private long id;
    private String message;
    private Date created;
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public Date getCreated() {
        return created;
    }
    public void setCreated(Date created) {
        this.created = created;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    private String author;

    public Comment() {
    }
    public Comment(long id, String message, String author, Date created) {
        this.id=id;
        this.message=message;
        this.created=new Date();
        this.author=author;
    }
}
```

CommentService



```
package com.syqliq.restxdemos.candour.service;

import java.util.ArrayList;

public class CommentService {

    private Map<Long, Message> messages = DatabaseClass.getMessages();

    public List<Comment> getAllComments(long messageId){
        Map<Long,Comment> comments = messages.get(messageId).getComments();
        return new ArrayList<Comment>(comments.values());
    }

    public Comment getComment(long messageId, long commentId) {
        Map<Long,Comment> comments = messages.get(messageId).getComments();
        return comments.get(commentId);
    }

    public Comment addComment(long messageId, Comment comment) {
        Map<Long,Comment> comments = messages.get(messageId).getComments();
        comment.setId(comments.size()+1);
        comments.put(comment.getId(), comment);
        return comment;
    }

    public Comment updateComment(long messageId, Comment comment) {
        Map<Long, Comment> comments = messages.get(messageId).getComments();
        if(comment.getId()<=0) {
            return null;
        }
        comments.put(comment.getId(), comment);
        return comment;
    }

    public Comment removeComment(long messageId, long commentId) {
        Map<Long, Comment> comments = messages.get(messageId).getComments();
        return comments.remove(commentId);
    }
}
```

CommentResource

```
@Path("/")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class CommentResource {

    private CommentService commentService = new CommentService();

    @GET
    public List<Comment> getAllComments(@PathParam("messageId") long messageId){
        return commentService.getAllComments(messageId);
    }

    @POST
    public Comment addMessage(@PathParam("messageId") long messageId, Comment comment) {
        return commentService.addComment(messageId, comment);
    }

    @PUT
    @Path("/{commentId}")
    public Comment updateMessage(@PathParam("messageId") long messageId,@PathParam("commentId") long id, Comment comment) {
        comment.setId(id);
        return commentService.updateComment(messageId, comment);
    }

    @DELETE
    @Path("/{commentId}")
    public void deleteComment(@PathParam("messageId") long messageId, @PathParam("commentId") long commentId) {
        commentService.removeComment(messageId, commentId);
    }

    @GET
    @Path("/{commentId}")
    public Comment getMessage(@PathParam("messageId") long messageId,@PathParam("commentId") long commentId) {
        return commentService.getComment(messageId, commentId);
    }
}
```



JERSEY

IX(H): PLAY BOOK: STATUS CODE AND LOCATION HEADER INFORMATION

MessageResource

- Adding additional information for status code and location header information using response object.

```
@POST  
public Response addMessage(Message message, @Context UriInfo uriInfo) throws URISyntaxException {  
    Message nuoMsg = messageService.addMessage(message);  
    String nuoId = String.valueOf(nuoMsg.getId());  
    URI msgUri = uriInfo.getAbsolutePathBuilder().path(nuoId).build();  
    return Response.created(msgUri)  
        .entity(nuoMsg)  
        .build();  
    //return messageService.addMessage(message);
```

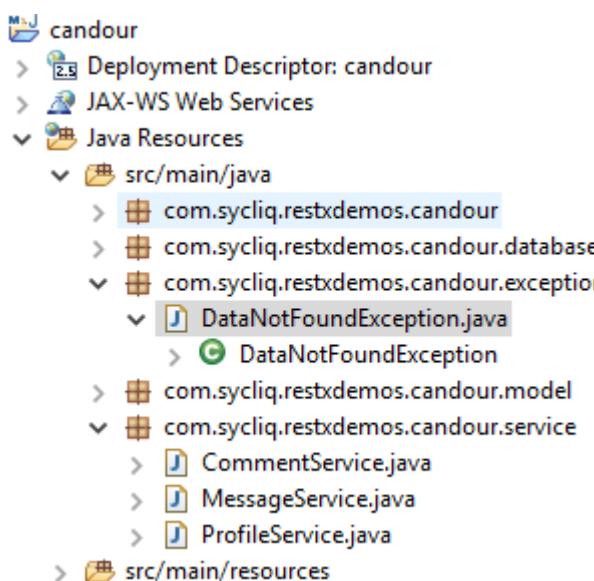


JERSEY

IX(I): PLAY BOOK: EXCEPTION HANDLING IN REST

DataNotFoundException

- Create a DataNotFoundException by extending RunTimeException



```
package com.sycliq.restxdemos.candour.exception;

public class DataNotFoundException extends RuntimeException {

    private static final long serialVersionUID = -217127812389123L;

    public DataNotFoundException(String message) {
        super(message);
    }
}
```

MESSAGESERVICE

```
public Message getMessage(long messageId) {
    //return messages.get(messageId);
    Message msg = messages.get(messageId);
    if(msg==null) {
        throw new DataNotFoundException("Message not found with Id:"+messageId+"NOT FOUND");
    }
    return msg;
}
```



ErrorMessage Domain Model

```
package com.syqliq.restxdemos.candour.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class ErrorMessage {
    private String errorMessage;
    private int errorCode;
    private String documentation;

    public ErrorMessage() {
    }

    public ErrorMessage(String errorMessage, int errorCode, String documentation) {
        super();
        this.errorMessage = errorMessage;
        this.errorCode=errorCode;
        this.documentation=documentation;
    }
    public String getErrorMessage() {
        return errorMessage;
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }

    public int getErrorCode() {
        return errorCode;
    }

    public void setErrorCode(int errorCode) {
        this.errorCode = errorCode;
    }

    public String getDocumentation() {
        return documentation;
    }

    public void setDocumentation(String documentation) {
        this.documentation = documentation;
    }
}
```



DataNotFoundExceptionMapper

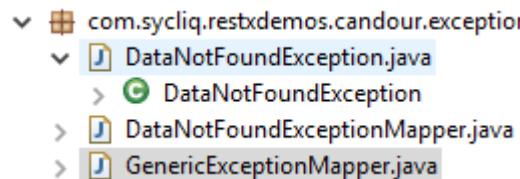
```
package com.syqliq.restxdemos.candour.exception;

import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;

import com.syqliq.restxdemos.candour.model.ErrorMessage;

@Provider
public class DataNotFoundExceptionMapper implements ExceptionMapper<DataNotFoundException> {

    @Override
    public Response toResponse(DataNotFoundException exception) {
        // TODO Auto-generated method stub
        ErrorMessage errMsg = new ErrorMessage(exception.getMessage(),404,"http://www.syqliq.com");
        return Response.status(Status.NOT_FOUND)
            .entity(errMsg)
            .build();
    }
}
```



GenericExceptionMapper

```
package com.syqliq.restxdemos.candour.exception;

import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;

import com.syqliq.restxdemos.candour.model.ErrorMessage;

@Provider
public class GenericExceptionMapper implements ExceptionMapper<Throwable> {

    @Override
    public Response toResponse(Throwable exception) {
        // TODO Auto-generated method stub
        ErrorMessage errorMsg= new ErrorMessage(exception.getMessage(),500,"http://www.syqliq.com");

        return Response.status(Status.INTERNAL_SERVER_ERROR)
            .entity(errorMsg)
            .build();
    }
}
```

Original Series

- ▼ com.syqliq.restxdemos.candour.exception
 - ↳ DataNotFoundException.java
 - > DataNotFoundException
 - > DataNotFoundExceptionMapper.java
 - > GenericExceptionMapper.java





SYED AWASE

X: JAVA WEB SERVICES: SUMMARY

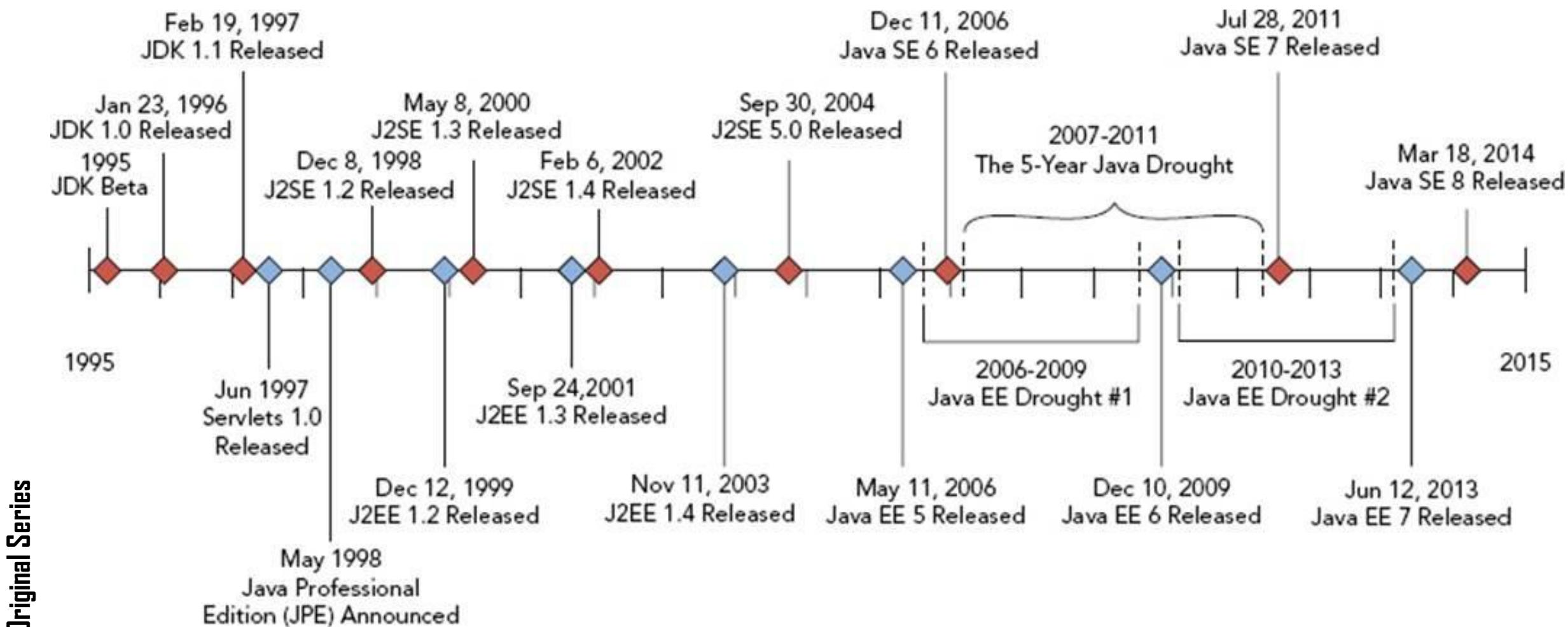


SOAP

- Simple Object Access Protocol
- SOAP cannot use REST because it is a protocol
- SOAP uses services interfaces to expose the business logic
- JAX-WS is the java API for SOAP web services
- SOAP defines standards to be strictly followed
- SOAP requires more bandwidth and resource than REST
- SOAP defines its own security
- SOAP permits XML data format only
- SOAP is a heavy weight operation
- SOAP is less preferred than REST

REST

- Representational State Transfer an Architectural Style
- REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP
- REST uses URI to expose business logic
- JAX-RS is the java API for RESTful web services
- REST does not define too much standards like SOAP
- REST requires less bandwidth and resource than SOAP
- RESTful web services inherits security measures from the underlying transport
- REST permits different data format such as plain text, HTML, XML, JSON etc.
- A Lightweight and suggested to use in lower band with scenarios such as accessing applications using mobile devices.

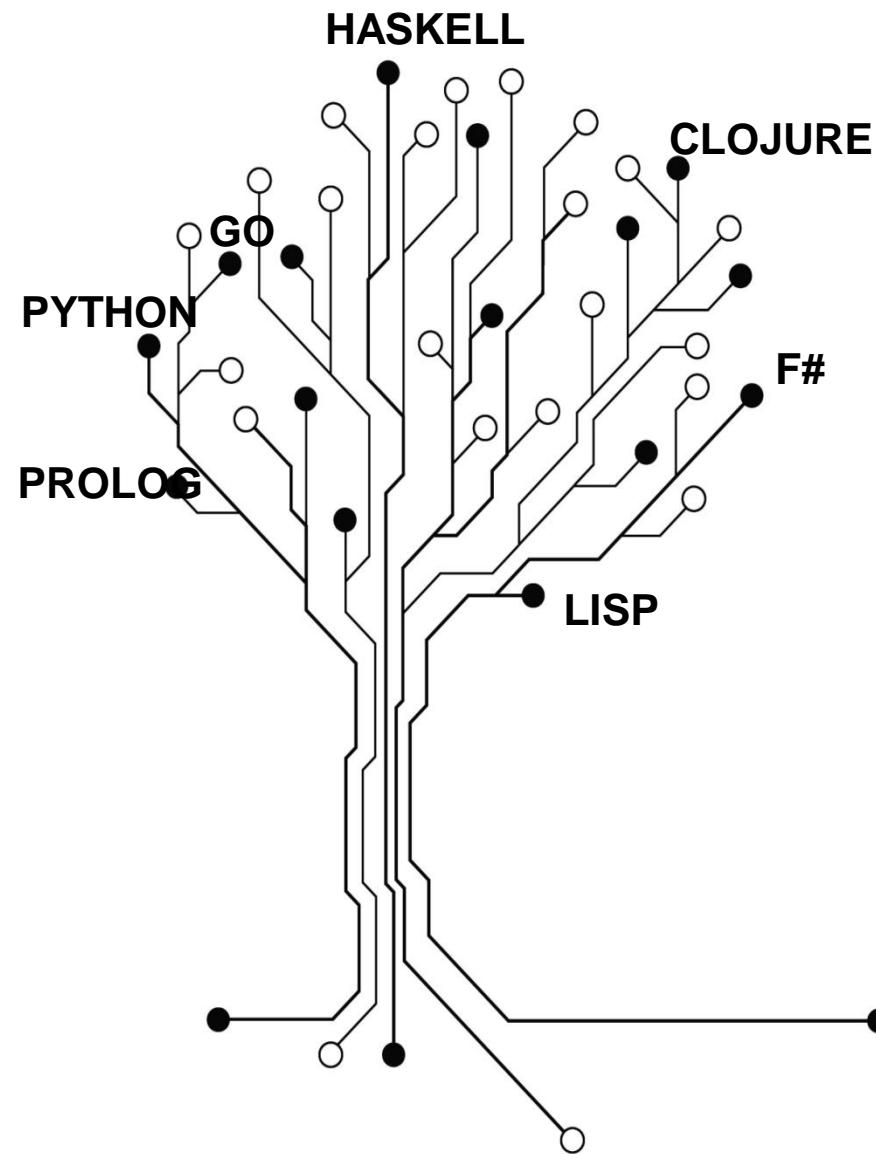


TPRI-SYCLIQ PROGRAMS OVERVIEW

EMPOWERING YOU

Artificial Intelligence

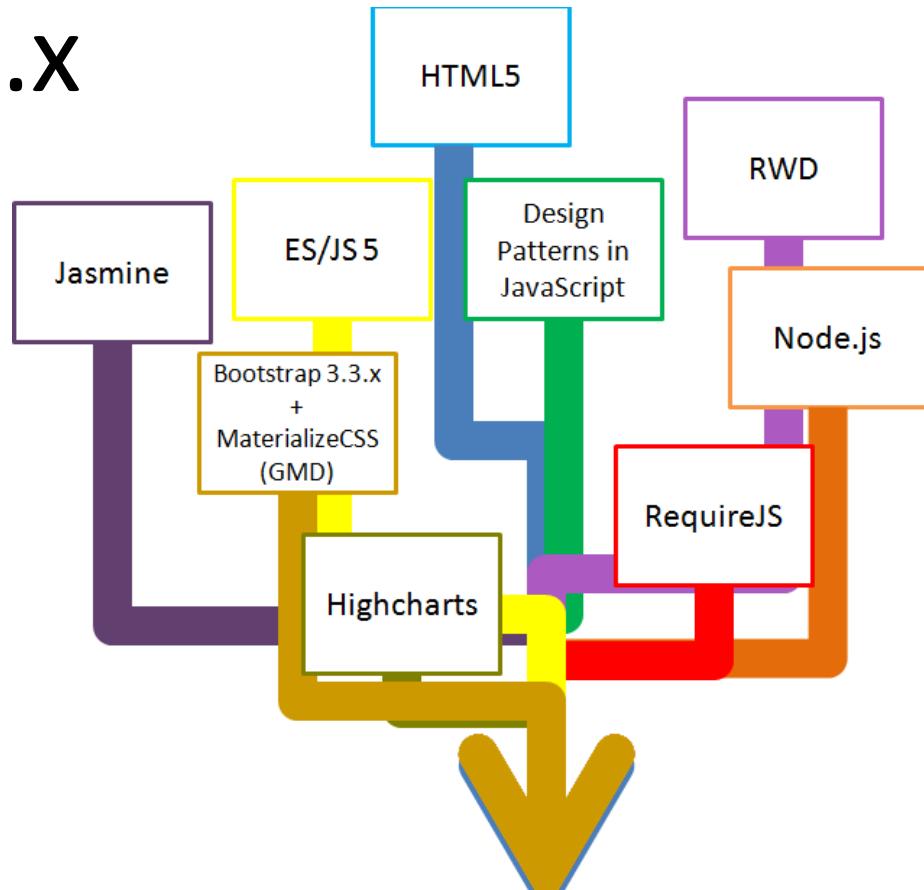
- We also train on AI Stack
- Reach out to us sak@sycliq.com or sak@territorialprescience.com
- www.territorialprescience.com
- www.sycliq.com
- +91.9035433124



Angular 2.x/Angular JS 1.5.x

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124



Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>



NOW OFFERING!

Code Focused Training

JAVASCRIPT FRAMEWORKS CODE DRIVEN CAPACITY BUILDING PROGRAM

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124

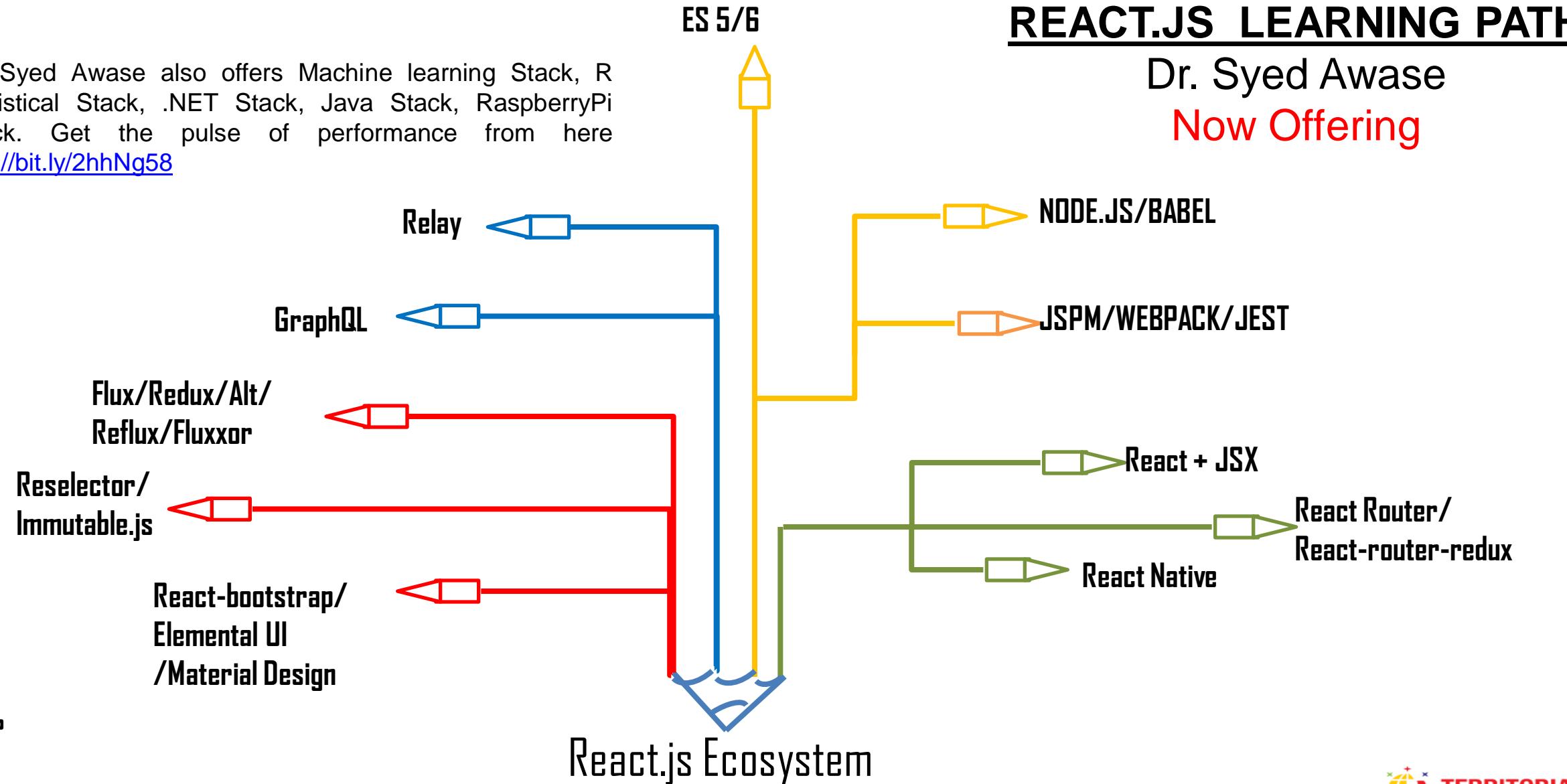


Original Series

Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack,
.NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>

© TPRI Syed Awase 2013-14 - Java Web Services Ground Up!

Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>



REACT.JS LEARNING PATH

Dr. Syed Awase
Now Offering

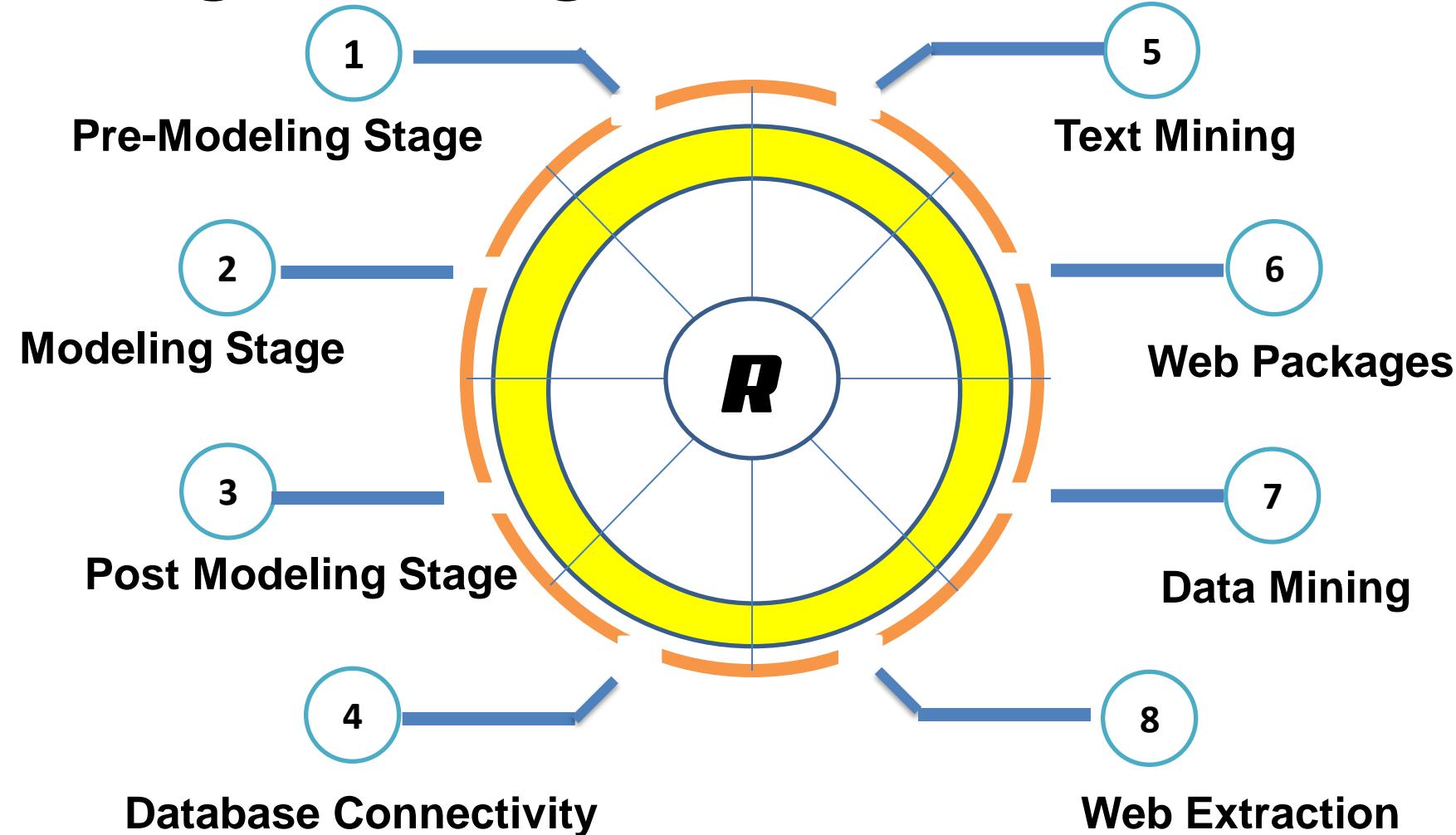


R-Statistical Programming

Dr. Syed Awase 2016 Session
Feedbacks: <http://bit.ly/2hhNg58>

Reach out to
sak@territorialprescience.com/
+91.9035433124

Original Series



Dr. Syed Awase also offers Machine learning Stack, R



©TPRI Syed Awase 2013-14 - Java

Statistical Stack,
.NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of

Web Services Ground Up!

performance from here <http://bit.ly/2hhNg58>

Thank You

We also provide Code Driven Open House Trainings : sak@territorialprescience.com or sak@sycliq.com

Original Series



- Java Technologies
- Core Java
 - Hibernate
 - Spring Framework
 - Play Framework
 - Hadoop
 - Groovy & Grails



- Microsoft Technologies
- C# Core
 - Entity Framework
 - MVC 5/6
 - Web Api
 - OWIN/KATANA
 - WCF
 - WPF



- Python
- Python
 - Django
 - Flask
 - Numpy
 - Scipy
 - Machine Learning



DATA SCIENCE

- R Statistical Programming
- Julia



- SQL and NoSQL
- Oracle
 - PostgreSQL
 - MSSQL
 - MongoDB
 - Neo4j
 - Redis
 - Firebase
 - Apache Cassandra



- Client-Side Frameworks
- Angular JS 1.5.x
 - Angular 2.4.x
 - React JS
 - KnockOut JS
 - VueJS
 - Backbone JS
 - EMBER JS
 - Hapi JS
 - METEORJS
 - MEANJS
 - Coffeescript
 - Dart



- Others
- LISP
 - CLOJURE
 - RUST
 - GO
 - Raspberry PI
 - Coming Soon
 - PHP
 - RoboticOS