**What is Web Service?**

Web Service is a client – server application that communicate over the network through HTTP protocol. Web Service provides a Standard means of interoperation between software applications running on the heterogenous platforms and frameworks.

**What is Rest Web Service?**

Representational State Transfer is an architectural style for developing applications that can be accessed over the network. REST is a stateless producer-consumer architecture where web services are producers which can be identified by the URI’s and consume application can use the HTTP methods to invoke RESTfull Web Services.

**Advantages of REST Web Services?**

1. REST is a light weight protocol.

2. Supports multi technologies for data transfer like text, JSON, xml, images etc.

3. No contract is defined between server and client so loosely coupled implementation.

4. REST methods can be tested over web.

**Disadvantages?**

1. since there is no contract it has to be communications through documents or emails.

2. session can’t be maintained

3. REST has no implicit security so we have to define the security.

**JAX-RS API?**

It is the java API for creating REST web services. It uses annotations to simplify the development of web services. As JAX-RS is part of JDK no need to include anything to use those annotations.

Two major implementations are:

1. Jersey.

2. RESTEasy.

What is SOAP Web Services?

Simple Object Access Protocol for exchanging XML-based messages over the network using application protocols like http. SOAP message comprises of a SOAP envelope. The envelope can be broken into a header and a body. Header contains context related definitions like security while the body contains actual message.

Advantages?

WS security: Soap defines its own security known as WS security.

Transport protocol neutral.

**Security for REST web services?**

WE can secure web services using one of the following methods

1. Updating the web.xml to define security configuration.

<security-constraint>

2. using Javax.ws.rs.core.SecurityContext interface to implement security programmatically.

3. using annotations like DeclareRoll, DenyAll, PermitAll, RolesAllowed etc.

4. using Jerssy OAuth libraries to sign and verify users.

**Various ways to test REST Web Services?**

Browser, programs, SOAP UI, Postman extension for chrome, Poster for firefox.

**SOAP or Simple Object Access Protocol:**

a) SOAP is a protocol for exchanging XML-based messages over the network using application protocols like http.

b) SOAP message comprises of a SOAP envelope. The envelope can be broken into a header and a body. Header contains context related definitions like security while the body contains actual message.

**WSDl:**

Web Service Descriptive Language

a) WSDL is a XML document which is used to describe a web service.

b) A WSDL completely describes what web service exposes, what parameter it expects and structure of output it returns, location of web service.

c) A WSDL contains definition, types, messages, portType, binding and service.

Definition: Root element of all WSDL documents. It defines the name of the web service.

Types: Data types in the form of xml schemas used in messages.

Messages: Arguments to be mapped to a method invocation. Data elements for each operation.

PortTypes: Operations that can be performed and messages involved.

Binding: provides details how porttype operations are transmitted over the network.

Service: Ports supported by webservices and location of wsdl service.

**Exception handling in REST Web Service?**

There are a few different ways to handle exception in Spring controller.

* Using ResponseEntity and HttpStatus codes
* Using @ResponseStatus on the custom exception class
* Using custom method to handle error on the controller (@ExceptionHandler and @ResponseStatus).
* Return error representation instead of default html error page

Apache CXF - contract first/top-down:

a) Define the data types in XSD

b) Define the WSDL

c) Add CXF dependencies (cxf-rt-frontend-jaxws, cxf-rt-transports-http) in pom.xml (MAVEN)

d) Use Wsdl2Java tool generate artifacts

e) Code the web service methods in the generated class

f) Declare the CXF servlet in web.xml

g) Publish service endpoints via a spring based configuration file.

<jaxws:endpoint id="bookShelfService" implementor="com.aranin.weblog4j.services.BookShelfServiceImpl" address="/bookshelfservice" />

h) Build the WAR

i) After the webapp is deployed, the web service (WSDL) is available via the URL

Apache CXF - contract - last/bottom-up:

a) Add CXF dependencies (cxf-rt-frontend-jaxws, cxf-rt-transports-http) in pom.xml (MAVEN )

b) Creating a Service Endpoint Interface (SEI)

- @WebService – This is an annotation JAXWS library. It turns a normal POJO into a webservice.

- @WebMethod – This annotation is optional and is mainly used to provide a name attribute to the public method in wsdl.

c) Implementing the service class

- @WebService(endpointInterface = "service interface", serviceName = "servicename")

d) CXF uses JAXB as default data binding component.

e) Publish service endpoints via a spring based configuration file.

<jaxws:endpoint id="bookShelfService" implementor="com.aranin.weblog4j.services.BookShelfServiceImpl" address="/bookshelfservice" />

f) Need to wire spring and CXF through web.xml

<servlet-class>

org.apache.cxf.transport.servlet.CXFServlet

</servlet-class>