**Web Services** – web of services

IOS – Swift programing language

Interoperability is the key (Independent of platform/applications)

DevOps – introduce various techniques

SOA – service oriented applications

A Web-Service is a Software system designed to support interoperable machine-to-machine interface over a network

RPC – Same platform and architecture should be same

CORBA – not universally accepted W3C, IOP only few programming languages accept

RMI –

**NEED FOR WEB-SERVICES: Universally accepted, distributed, interoperable and platform neutral protocol**

**WSDL – Web Services Definition Language**

An XML based Interface (kind of defining contract to which all classes has to abide/adhere)

Current version WSDL 2.0 (W3C)

1. **SOAP – Simple Object Accessible Protocol**

An XML based protocol/requests from service consumer (User) to Service provider (repository)

Ex.: Weather, time-zone etc.

HTTP + XML = SOAP

SOAP Request -

SOAP Response -

1. **RESTful – Representational State Transfer**

Lightweight and less formal approach

**WADL** **– Web Application Description Language**

Both XML and JSON web service requests

**JSON – Java Script Object Notation**

Parsing is easy in JSON because only keys are passed as parameters

**Implementations:**

**JAX-WS** – Java API for web services for SOAP

Apache-CXF - Implementer of JAX-WS part of Java SE

**JAX-RS** – Supports web services for REST

Jersey – implementer of JAX-RS not part of Java SE

XML –

XSD – Schema/vocabulary for XML, for defining the rules

**XML is case sensitive and XML parser are non lenient** (tags has to be closed)

Attribute values has to be started and ended with double or single quotes (“ & ‘)

**Markup/meta language - Language meant to describe other language**

Common data – Attributes

Elements – opp of att

Un-parse character data:

For O’Hara, for ‘ it is **&apos**

So O**&apo;**Hara

1. Non-Validating Parser: only to check XML

Usage-

<![CDATA[ ….unparsed data ] this will ignore the rules and proceed

1. Validating Parser: validate against the schema

Struts and DOM – low level APIs and lot of iterations required, hence we widely use -

**JAXB – Java Architecture for XML Binding**

**Directly read/write the XML file**

**or OXM Object to XML Mapping**

**Marshaling and un-marshaling**

Converting Objects to XML and XML to Objects

**Download and install Java 1.8 latest version**

**Download below 3 archives and unzip the folders**

**spring-tool-suite-4-4.2.2.RELEASE-e4.11.0-win32.win32.x86\_64.zip**

**apache-cxf-3.3.2.zip**

**apache-tomcat-9.0.21-windows-x64.zip**

**Launch --- SpringToolSuite4.exe**

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

**XML Schema and XML file generation steps:**

Create new workspace, create New-Other-General-Project

Create New-Other-General-XML Schema File (orders.xsd)

Inside Source – Add the file contents

Create New-Other-General-XML File(orders.xml) – Next

* Select - Create XML File from an XML Schema File – Next

Browse the created XSD file (orders.xsd)

Update the contents

Validate both the files

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

**Marshalling and Un-marshalling**

Create New-Other-Maven Project

Select Create simple project () box – Next

Group ID and Artifact ID input (orders-jaxb)

Input any Name (my-jaxb)

Input Description (Marshalling and Un-marshalling) – Next

Wait for 5-10 min for the Project jars and contents to load

Pom.XML will be auto generated

Clear the contents of Pom.xml and add contents shared by trainer (Google classroom)

---- Update paths and folders name --------- (compare both files for the diff)

Ignore the <execution> error in pom.xml and proceed ahead

Create new Folder (schema) under src/main/resources

Copy orders.xsd file to this location

Right click on orders-jaxb -- Maven – Update Project

Refresh and check for Marven Dependencies creation and contents inside the structure

Create new file (orders.xjb) for custom java file calls

Clear the contents of orders.xjb and add contents shared by trainer (Google classroom)

---- Create new class (DateConverter.java) under src/main/java – com.myapp.ws.converter

Add the code shared by trainer (Google classroom)

Import the required packages

For ConverterException, quickFix – create class ConverterException

Update Super class as java.lang.RuntimeException

Check the box Constructors from Superclass and Finish

Project – Clean

orders-jaxb – Righ click – Run as – Run Configurations

Select New launch configuration

Base Directory – Workspace and browse select the Maven Project (orders-jaxb)

Goals - input - clean compile – click on Run

Check for Build successful in console

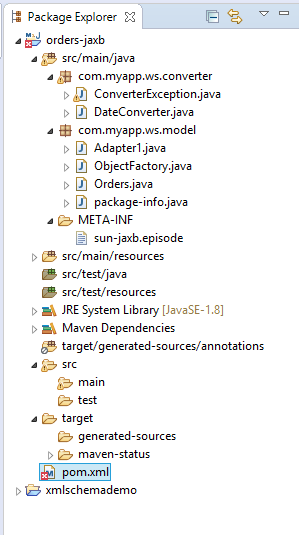
Refresh orders-jaxb and check for the 4 java classes create under : package com.myapp.ws.model

Adapter1.java

ObjectFactory.java

Orders.java

Package-info.java



----------------------------------------------------------Day 2 18-June-2019------------------------------------------------------

In Orders.java

**After public static class Order**

**implements Equals, HashCode, ToString**

**Ctrl + Space – create new Constructor**

**Right click – Source - Generate Constructors using Fields**

**Check – omit calls to default constructor – save the class**

**Marshalling:**

Create new class under Model package and change

Package – marshaller

Name – TestMarshaller

Check box - Public static void string

TestMarshaller.java – Run As – Java Application – check for the XML as syso output

**UnMarshalling:**

Create new class under Model package and change

Package – unmarshaller

Name – TestUnMarshaller

Check box - Public static void string

TestUnMarshaller.java – Run As – Java Application – check for the XML as syso output

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

**SOAP – Simple Object Access Protocol**

SOAP Part – Header and Body of XML

HTTP, SMTP etc.

Attachment Part – XML or non-XML (ex.: certificates)

**SOAP Template:**

<soapenv : Envelop>

<soapenv : Header> (optional, for credentials and others)

< / >

<soapenv : Body> (required)

<payload>

</ >

**WSDL:**

**Root element is <definitions>**

**<types> : for XML Schema ex.: Data types etc.**

**<message> : parameters.. What Request msg and what Response msg**

**<port Type> : define supported operations**

**<binding> : what protocol to be used like http/amqp/smtp**

**<service> : identity the end point access/location**

**Difference b/w WSDL 1.0 and WSDL 2.0**

**Message is removed and Port type is changed to Interface**

JAX-WS – Implementers like --- Metro, **apache-cxf**, AXIS

**Two options/approaches to enable Web services:**

Top down

Schema WSDL and generate Java classes

Bottom up

From Java and Interfaces generate WSDL and schema

##############################################################################33

**In Spring Tool Suit or in Eclipse Oxygen:**

Windows – Preference

1. Server – Runtime Environment – Add – Select Apache Tomcat V9.0 – SELECT

Browse and select the downloaded file “apache-tomcat-9.0.21”

Change JRE to working version 1.8 – FINISH

1. Web Services – server & Runtime – server run time - select Tomcat 9.0
2. Web service Runtime – select – Apache CFX 2.x - Apply
3. CXF 2.x preference – CFX Runtime – browse and select apache cft downloaded folder
4. Check box the browsed file

File – New – Dynamic Web Project – (orders-webservices-topdown)

Configuration - <custom> - modify – CFX 2.0 …. OK – NEXT – Context root – itc – tick Generate…. – Next – CFX Runtime should be selected by default – FINISH

Wait for 5-10 mins for the classes and jars to load

WebContent – New Folder – wsdl

Copy orders.xsd and orders.xjb to this new folder (wsdl)

Create new Package under Java Resources - src

Under wsdl new folder – New - WSDL file (orders.wsdl) - Next

Update Target namespace - <http://myapp.com/orders/> - Finish

Update below files as in Repo::

orders.wsdl and orders.xsd

New – Other – select: Web Service

Web Service Type – change to Top Down

Service Definition – Browse – Browse – select: orders.wsdl

Change Bar to Deploy Service only – Next

Change package name to - com.myapp.ws

Select service name orders

Binding File: Browse and add renamed orders.xml (orders.xjb to orders.xml) using **browse file system** option:

**eclipse-workspace**\orders-webservices-topdown\WebContent\wsdl\ orders.xml

**FINISH**

Refresh and check for the 9 java classes created under com.myapp.ws Package

Cxf-beans.xml – remove 3 imports

Replace id=*"orders"* implementor=*"com.myapp.ws.OrdersImpl" as*

id=*"orders"* implementor=*"com.myapp.ws.OrdersSOAPImpl"*

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

Deploy services:

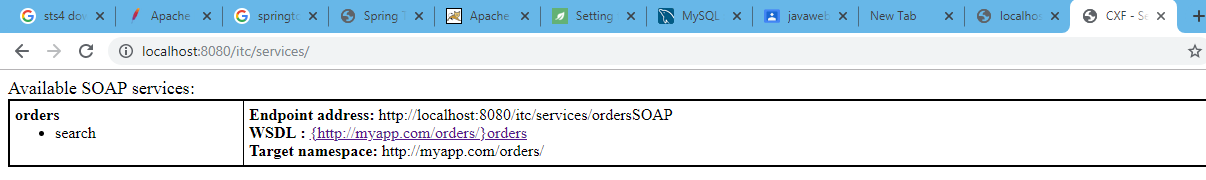
Bottom Pane, Servers – By Default no servers will be listed

Click and Add new server, Tomcat v9.0

Wait for some time…

Check the console for the exceptions if any…

Chrome - <http://localhost:8080/itc/services/>



-------------------------------------------------------------Day3 19-June-2019-------------------------------------------------

Chrome Settings – More Tools – Extension - Open Chrome Web Store – Search for SOAPClient

Boomerang should be added as chrome extension and restart chrome

Open Boomerang Ext in chrome and create service

WSDL URL - <http://localhost:8080/itc/services/ordersSOAP?wsdl>

Service Name – OrdersService – CREATE

Under Search – Create Request

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

POSTMAN – same function as above

RAW Tab to be updated in Postman after hitting the WSDL URL - <http://localhost:8080/itc/services/ordersSOAP?wsdl>

<x:Envelope xmlns:x="<http://schemas.xmlsoap.org/soap/envelope/>" xmlns:ord="<http://myapp.com/orders/>">  
<x:Header/>  
<x:Body>  
<ord:ProductName>  
<ord:productName></ord:productName>  
</ord:ProductName>  
</x:Body>  
</x:Envelope>

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

**FURTHER IMPLEMENTATION**

Order.java

In Orders.java

**After public** **class** Order {**Ctrl + Space – create new Constructor**

**Right click – Source - Generate Constructors using Fields**

**Check – omit calls to default constructor – save the class**

Create New Class:

com.myapp.ws.reporsitory

OrderRepository

Update Code in OrderRepository.java

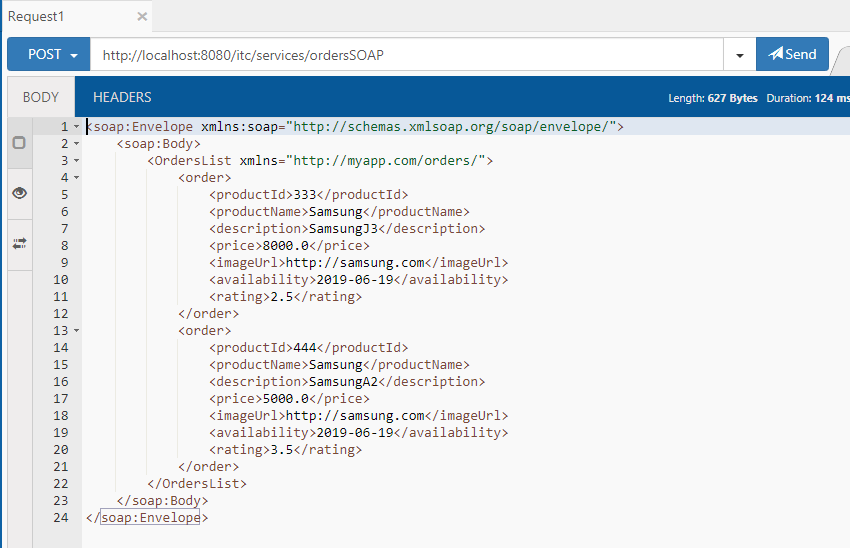
Update Code in OrdersSOAPImpl.java

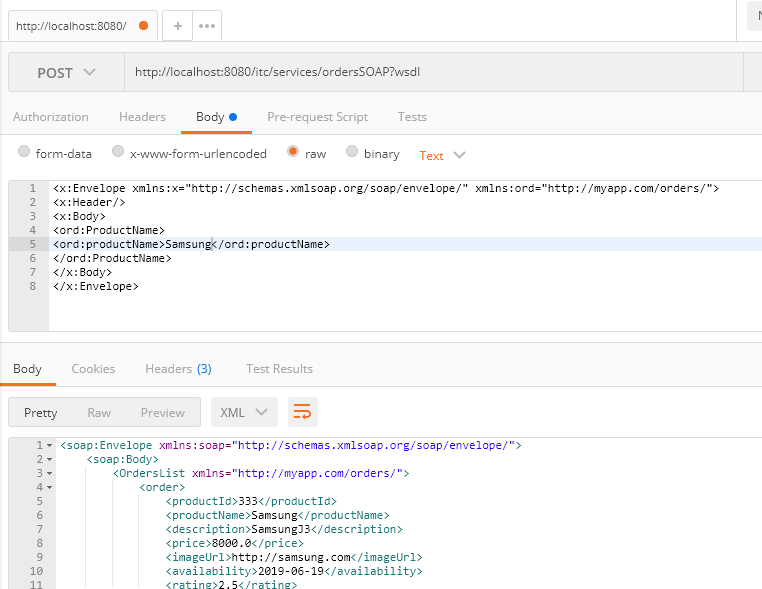
Restart server, check in Boomerang/Postman by proving the Product name (updated in OrderRepository.java) in the request XML

**Example**:

<ord:productName>Samsung</ord:productName>

Output should be an XML format as below after sending the request:

**Boomerang**:

**Postman**:

**How to consume Web Services:**

**Very Important – Tomcat Server should be started**

New Project – Dynamic Web Project

Name: orders-webservice-topdown-client

Configuration - <custom> - modify – CFX 2.0 …. OK – NEXT – Context root – wsclient – tick Generate…. – Next – CFX Runtime should be selected by default – FINISH

Wait for 5-10 mins for the classes and jars to load

Copy the package from previous project to current Project

Copy orders.xml from previous project - wsdl folder

Create new Folder in current project – WebContent as “xjc” and paste the order.xml

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

New – Other – Web Services Client

Service Definition : <http://localhost:8080/itc/services/ordersSOAP?wsdl>

Custom Bar – set to **Assemble Client** only – Next

Package Name - com.myapp.ws

Binding Files – Add - : Browse and add renamed orders.xml (orders.xjb to orders.xml) using **browse Workspace** option:

**C:\Users\12888\eclipse-workspace\orders-webservice-topdown-client\WebContent\xjc\orders.xml**

Next – uncheck Generate Implementation

**FINISH**

Refresh and check for the 8 java classes created under **com.myapp.ws** Package

Update Orders\_OrdersSOAP\_Client.java:

\_search\_productName.setProductName("productName"); to

\_search\_productName.setProductName("Apple");

Order.java:

After **public** **void** setRating(**double** value) {

Right click – Source – Generate toString()

Change Code Style – StringBuilder/StringBuffer – Generate

Go To - Orders\_OrdersSOAP\_Client.java:

Right Click – Run As – Java Application

Check for the Product details matching the Product name updated in the same Java file in the console.

JAX-WS

2 methods of calling SOAP web services

Inbuilt tool wsimport (something like a command line tool) however this is not used instead we use Apache CFX both for server side and client side. Exlipse or IDE’s wizard uses these wsimports to do the same task.

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

**Bottom-Up Jax-WS Web Services:**

**Chrome:** <https://start.spring.io/>

Group: bottom-up-orders-webservices

Dependencies, search web and add

spring web starter

spring web services

Spring Data JPA

MySQL Driver

GENERATE THE PROJECT

Extract the downloaded content (demo.zip)

In Spring tool suit 4x:

File – Import – Maven – Existing maven project – Browse and select the extracted folder

Wait for 5-10 mins for the files to import

Under demo/src/main/resources – New Folder – schema – Add 2 files

New Package under demo/src/main/java as com.myapp.ws.converter

Add 2 java files from previous project – ConerterException anmd DateConverter

Update pom.xml

Update Java File – Order.java

New Interface – Package: com.myapp.ws.reporsitory

Name: OrdersRepository – FINISH

Update OrdersRepository.java

New Interface – Package: **com.myapp.ws.service**

Name: OrdersService

Update OrdersService.java

New Class – Package: com.myapp.ws.service

Name: OrdersServiceImpl

Update OrdersServiceImpl.java

com.myapp.ws.service.OrdersService

New Class - Package: com.myapp.ws.endpoint

Name: OrdersEndPoint

Update OrdersEndPoint.java

Refactor – Rename - Package bottomuporderwebservices to com.myapp.ws

Replace the code in BottomUpOrdersWebserviceApplication.java from classroom shared

Update pom.xml contents (has MySQL configurations)

Update application.properties (has MySQL configurations) as below:

spring.datasource.url=jdbc:mysql://10.6.140.199/sakila

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

Inside BottomUpOrdersWebserviceApplication.java

Run As – Java Application

<http://localhost:8080/services/orders.wsdl>

---------------------------------------------------------Day 4 20-June-2019--------------------------------------------------

Boomerang – New Service

<http://localhost:8080/services/orders.wsdl>

bottomupordersservice

Add Tag in Orders.xsd to add a new record

<!-- To Add a new Record -->

Update orders.xsd with required functions/methods

Save and update Maven project/run as maven build, clean compile

Check for the new java class generations, Add….,java (2 files)

Update OrdersEndPoint.java to add 2 more additional methods (Add methods)

Modify the annotations @Entity and others if they are reverted to previous state due to update project

Inside BottomUpOrdersWebserviceApplication.java

Run As – Java Application/Spring Boot application – wait for the start time in secs

<http://localhost:8080/services/orders.wsdl>

Boomerang – root folder update to get the new method addOrder

Create Request – Input all product details to add records (ID since sequential, not required.. tag can be removed)

Again search request and check for the new record creation(s)

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

**RESTful Web Services:**

**RE**presentational **S**tate **T**ransfer

Hypermedia – Anything can be used like XML, JSON

Implementer – JAX-RS

**Constraints**:

Stateless Communication – should contain all info needed to understand the request

Uniform Interface -

Layered System -

Hypermedia Based –

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

Abstraction known as “resource”

Resource is identified by URIs – unique

Resource - noun or a thing

We can nounfy the resources

**Data Input:**

* Matrix Parameters - <http://host/collection;mname=value>

through parameters like **;**

* URL Path Parameters – [c/{id}](http://host/collection/%7bid%7d)

through parameters like **/**

* Query Parameter - <http://host/collection?filter=A&filter=B>

through Query string like **?**

* Resource Representation (HTTP body)
* Headers

Examples:

<https://jsonplaceholder.typicode.com/>

Expand: /users

Query Parameter-

<https://jsonplaceholder.typicode.com/users?username=Bret>

URL Path Parameters-

<https://jsonplaceholder.typicode.com/users/1>

\*\* SWAGGER - API Documentation (for resources) alike schema in SOAP based web services

JSON – A web browser (Javascript)

100 services – information

200 – Success, client status/errors

300 – Redirection

400 – Fatal errors

500 – Server errors

REST:

300 – Multiple choices

30 – Moved permanently

303 – See other

304 –

GET – Read only & Idempotent (repeated calls)

PUT – Idempotent, Update operation

DELETE – Idempotent

POST – Non-Idempotent, for Addition of records

<PATCH> - Partial update

**HATEOS** – hypermedia engine of the states

Java Implementer – **JAX-RS,** Sun OOTB JERSEY - not much used (implementation of JAX-RS), apache CFX, RESTless, RESTeasy etc.

**JAX-RS Reference Implementation**

Jersey supports the implementation of RESTful web services

JAX-RS (JSR-311) implementation:

@produces - marshalling

@consumes – un- marshalling

----------

@Path – to map the Resource name

#########################################################3

<https://start.spring.io/>

Group ID & Artifact ID: rest-ws-demo

Options, Package Name: com.myapp.ws.rest

Dependencies: Spring Web Server, Spring Data JPA, MySQL Data, Jersey

Additional for security, Spring Security, OAuth2 client

Generate the Project and unzip

Extract the downloaded content (rest-ws-demo.zip)

In Spring tool suit 4x:

File – Import – Maven – Existing maven project – Browse and select the extracted folder

Wait for 5-10 mins for the files to import

Copy previous project (Bottom-up) packages

com.myapp.ws.converter

com.myapp.ws.model

com.myapp.ws.reporsitory

com.myapp.ws.service

Create New:

com.myapp.ws.rest.configuration

OrdersRestEndPoint

Unselect all 3 check boxes

AND

com.myapp.ws.rest.configuration

JerseyConfig

<http://localhost:8181/orders/application.wadl>

<http://localhost:8181/orders/orders/redmi>

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

Implement Other Operations

**POST**

Update Java classes

JerseyConfig

OrdersRestEndpoint

<http://localhost:8181/orders/api>

Body-

{

"productId": 3,

"productName": "REDmi",

"description": "REDmiNote7PRO",

"price": 7000,

"imageUrl": "https://REDmi.com",

"availability": "2019-06-20",

"rating": 4.25

}

<http://localhost:8181/orders/api/redmi>

-----------OTHER OPERATIONS------------------

Find-All, update and Delete operations:

Update below Java files,

OrdersService

OrdersServiceImpl

OrdersRestEndpoint

**Find-All**

<http://localhost:8181/orders/api>

This will list all the Products created

Update

<http://localhost:8181/orders/api>

{

"productId": 4,

"productName": "Redmi",

"description": "Redmi\_UPDATE",

"price": 99,

"imageUrl": "https://REDmi.com",

"availability": "2019-06-18",

"rating": 4.25

}

**Delete**

<http://localhost:8181/orders/api/6>

\*\*\* **6** is the Product ID

-----------------------------------Day4 21-June-2019--------------------------------------------

**SWAGGER Configuration – Securing the Web Services**

Swagger is a api documentation tool typically used for Restful WS

New class – SwaggerBeanConfigurer.java

Update JerseyConfig.java code

RestWsDemoApplication.java

Update – target – Pom.xml

Update Maven Project

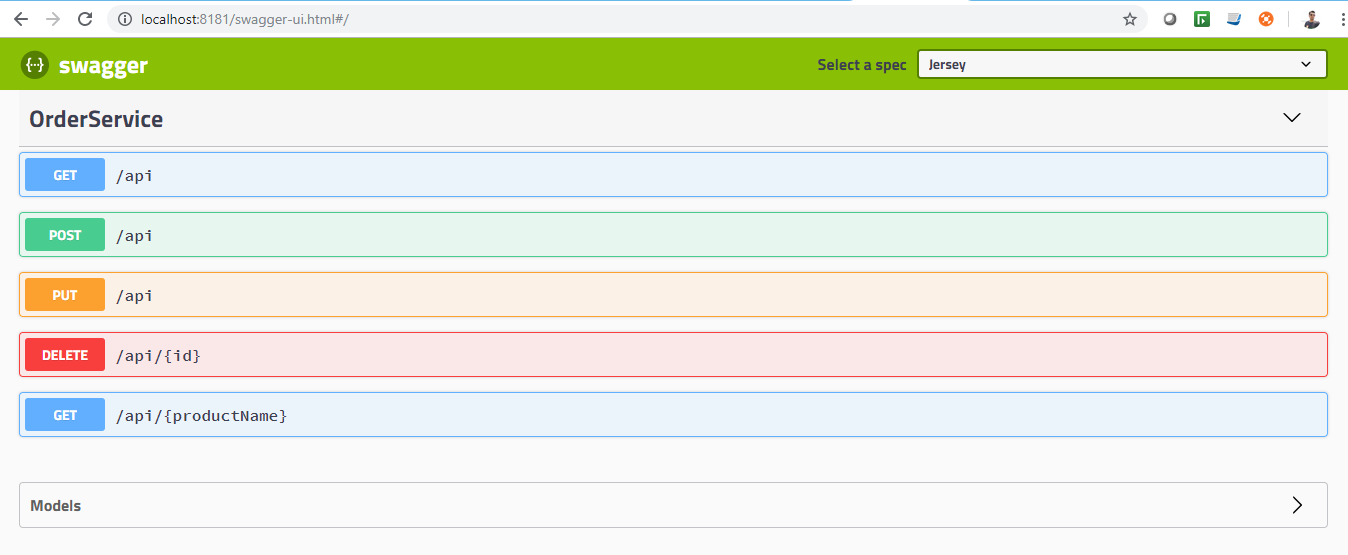
Update application.properties

## Swagger Config property

spring.jersey.application-path=/api

Check the Swagger UI for all applicable operations:

Chrome - <http://localhost:8181/swagger-ui.html>



Update OrdersRestEndpoint.java and check below:

Chrome - <http://localhost:8181/api/application.wadl>

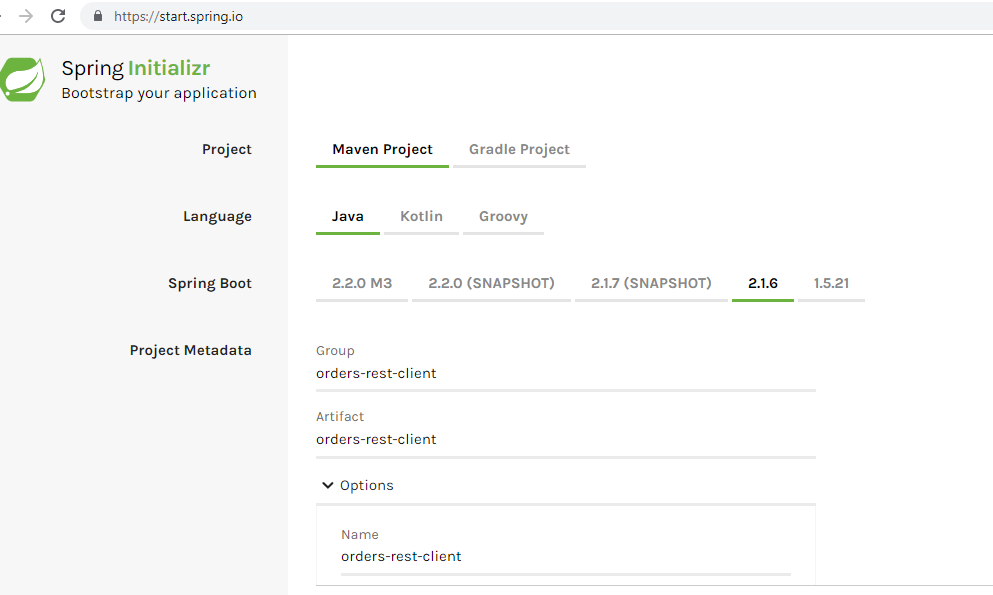
Boomerang - <http://localhost:8181/api/orderapi>

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

To Consume Web services:

<https://start.spring.io/>

Create new Project:



Group, Artifact and Name - orders-rest-client

Package - com.myapp.ws.rest

Dependencies:

Spring Web Starter

Jersey (for Client libraries)

Import existing Maven project in Eclipse/STS

Create new class,

OrdersRestClient.java – Update code

Update OrdersRestClientApplication.java

Update Maven Project or Run as – Run Config clean compile

Make sure previous project Java Application/Spring Boot App is up and running

Run this project OrdersRestClientApplication.java as Java Application/Spring Boot App is up and running

Check for the product details in console as updated in

SOAP Fault, exception handling

<S:Fault>

Top down approach

tags to be updated in the wsdl schema file

Bottom-up approach:

A new method to capture exceptions

<xs: complexType name = “Exception”>

<xs: sequence >

<xs: Element message = “Exception message “ >

<xs: /sequsence >

<xs: /complexType>

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

Web services Security:

SOAP supports – Transport-Level Confidentiality (HTTPS to supply) and Message-Level Confidentiality (WS severity end-to-end)

Restful – Only Transport-Level Confidentiality

**Asymmetric-Key-Cryptography**

**Example – RSA tokens**

Master and shared Key is shared b/w server and client during the initial handshake.

Encryption and Decryptions functions are used

**Session-Key-Cryptography** (widely used)

SSL server (session key) certification, Random pre-master key is created.

SSL – secured socket layer

**Examples – AES, EES**

Client connects to SSL server

Server Sends certificate with public key

Client created random key

Client uses server’s public key to encrypt

Client sends encrypted pre master key

Server decrypts premaster key

Server creates secret session key

Both use secret session key for further communications

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

**Javax.annotations.security common annotations**

JSR250 common annotations:

@RunAs

@RolesAllowed

@PermitAll

@DenyAll

######################################

**Enabling Basic Authentication:**

Add dependency in rest-ws-demo – Pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-oauth2-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Update Maven Project, RestWsDemoApplication.java - Run As java/spring boot app and check the Authentication screen in Boomerang while hitting the paths

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

**OAUTH 2.0**

**OAuth is not an Authentication its Authorization**

Example: Accessing IRCTC using Facebook login

FB sends **Access Tokens** to IRCTC to authorize user.

**JWT – JSON Web Tokens**

**Actors in OAUTH 2.0**

Resource Owner - User

Resource Server - IRCTC

Client -

AuthZ Server – FB

Client –> Auth Req

Auth Grant –> Client

Client –> Auth Grant

Access Token <- Client

Client –> Access Token

Client –> Access Token

Protected Resource –> Client

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

Copy paste previous project (rest-ws-demo) and rename as orders-rest-security-JAXRS

New File - application.yml

<http://localhost:8080/>

The link should redirect to gitbub/other website clientID and clientsecret provided in application.yml file, once logged in – Authorize user option should be displayed from to gitbub/other websites.

To configure github/google/other website google **“get github id and secret key**” and configure the settings as suggested.

---------------------------------------- END ----------------------------------------------------