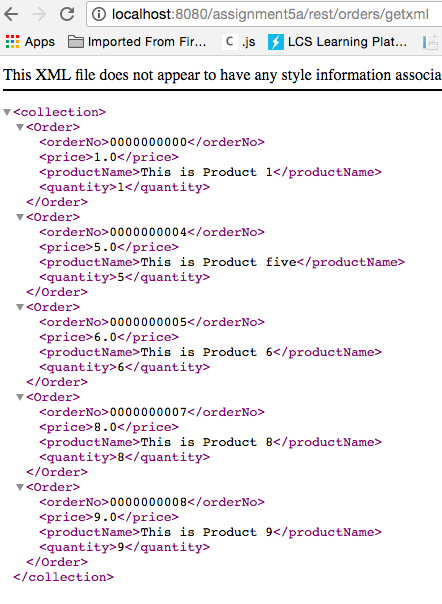
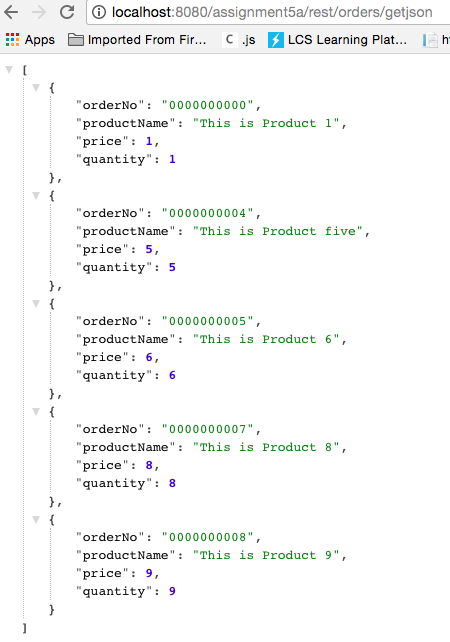
**Short Assignment/In-Class Activity #5 – Integration Services**

**Short Assignment 5a: Using JAX-RS to design and build a RESTful Web Service**

Objective:

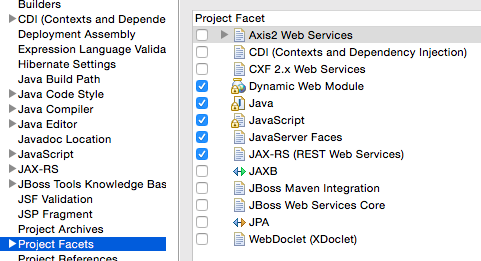
1. Design and develop a REST Based Service using JAX-RS.

In part 5a you will create a class that creates and returns a list of JSON and XML entries from the database.

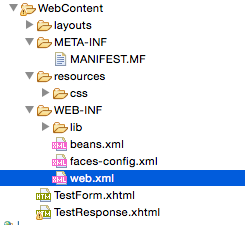


Activity Directions:

# Configure your JBOSS project to use Rest Services

Add and update the following components in the project:

* Project Configuration (add JAX-RS support):
  + Right click on the project and select the Properties menu option.
  + Navigate to the Project Facets.
  + Select the JAX-RS facet.



* Application Configuration (add Resteasy JAX-RS):
* Open the web.xml file
  + Add the following to the bottom of your web.xml file:

<!-- JAX-RS Servlet using RestEasy included in JBoss EAP -->

   <servlet-mapping>

        <servlet-name>Resteasy Servlet</servlet-name>

        <url-pattern>/rest/\*</url-pattern>

    </servlet-mapping>

    <context-param>

        <param-name>resteasy.scan</param-name>

        <param-value>true</param-value>

    </context-param>

    <context-param>

        <param-name>resteasy.servlet.mapping.prefix</param-name>

        <param-value>/rest</param-value>

    </context-param>

    <listener>

        <listener-class>

            org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap

            </listener-class>

    </listener>

    <servlet>

        <servlet-name>Resteasy Servlet</servlet-name>

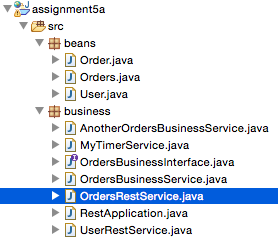
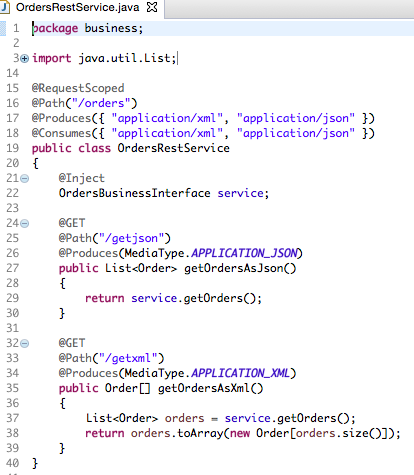
        <servlet-class>

            org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher

        </servlet-class>

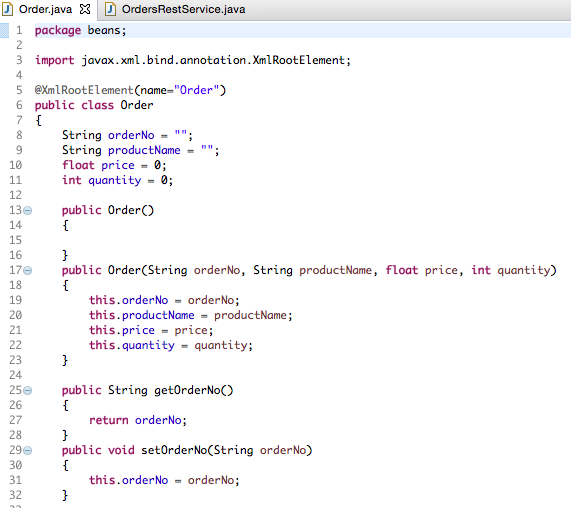
    </servlet>

## RESTful based Web Service:

* 1. Right click on the business package and select the New->JAX-RS Resource menu options.
  2. Enter a Name of *OrdersRestService* (this is the name of your web service class).
  3. Enter a Resource path of */orders*. This will become part of the URL in the final application.



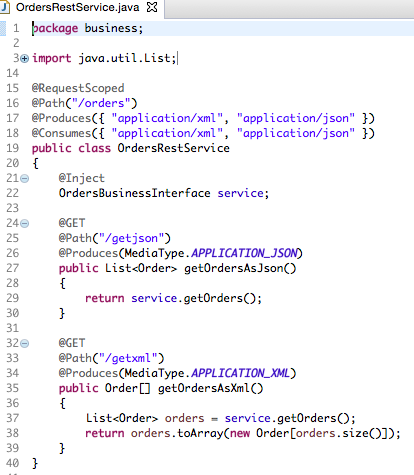
* 1. Click the Next button.
  2. Select the ‘Skip the JAX-RS Application create’ option. Click the Finish button.
  3. Inspect the annotations and code generated.
  4. Inject (use the @Inject annotation) the class scoped variable *service* of type **OrdersBusinessInterface**.
  5. Add the @XmlRootElement(“Order”) annotation to the ***Order*** class.
  6. Add a default constructor to the ***Order*** class.



* 1. Add Web Service methods to the **OrdersRestService** class.
     + Add a public method named *getOrdersAsJson()* that returns a List<Order>.
     + Add the @GET, @Path(“/getjson”), and @Produces(MediaType.APPLICATION\_JSON) annotations to the *getOrdersAsJson()* method.

This notation will create the last portion of the data access URL



* + - In the method implementation call the *service* variables getOrders() method and return the *service* method call as a return value for the method implementation.
    - Add a public method named *getOrdersAsXml()* that returns a Order[].
    - Add the @GET, @Path(“/getxml”), and @Produces(MediaType.APPLICATION\_XML) annotations to the *getOrdersAsXml()* method.



* + - In the method implementation call the *service* variables getOrders() method, convert the ArrayList List<Order> returned from the service call to an Order[] array, and return the array as a return value for the method implementation.

## Run the Code

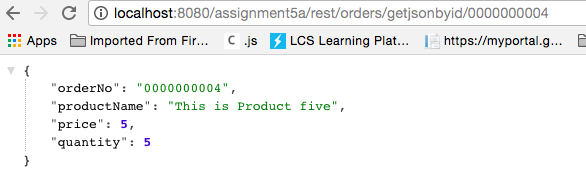
Deploy the code and execute each of the Web Service calls using your desktop browser.

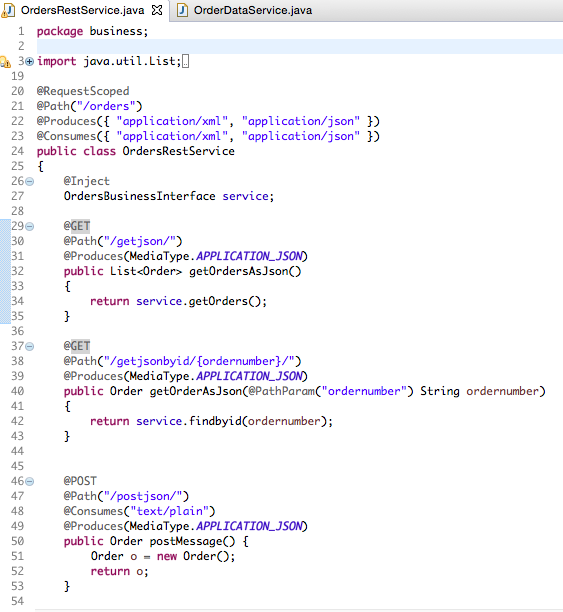
<http://localhost:8080/assignment5a/rest/orders/getjson> (*assignment* is your project name)

<http://localhost:8080/assignment5a/rest/orders/getxml> (*assignment* is your project name)

NOTE: for the XML output you should test in the Chrome browser. Take a screenshots for the JSON and XML methods. The Orders List of size 10 should be displayed in the browser with XML and JSON. Note: you can format the JSON output to improve the readability by copying the JSON output from your browser into a tool provided at jsonprettyprint.com.

# Create a second Rest service that searched an order by its id number.

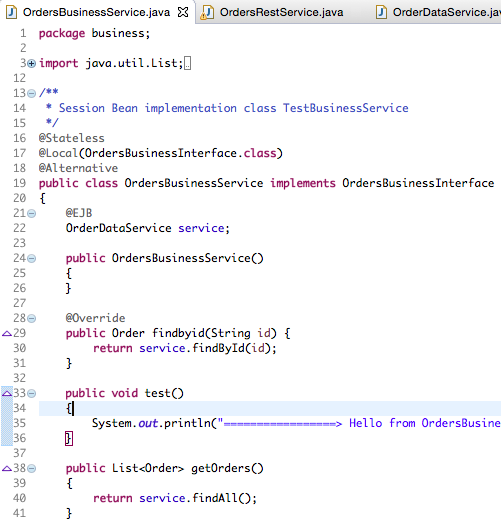




Here is the new code for the JSON service inside **OrdersRestService**

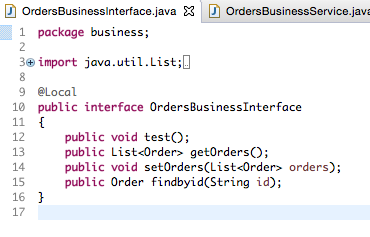
It appears that the service.findbyid(ordernumber) method generates nothing. The database service has not yet been implemented.

Go through the following files and create a findbyid method.



OrderBusinessService needs some code changes…

I changed OrderBusinessInterface to use a string parameter type instead of int.



Implement the findById(String id) method in OrderDataService…



**Short Assignment 5b: Using JAX-WS to design and build a SOAP Web Service**

Objective:

1. Design and develop a SOAP Based Service using JAX-WS.

Activity Directions:

Study the Web Services example code from the reading for this Topic. This activity will add code to the previous activity.

Add and update the following components in the project:

SOAP based Web Service v1:

1. Right click on the business package and select the New->Other->Web Services->Simple Web Service menu options.
2. Select the current project from Dynamic web project dropdown list.
3. Enter a Name of *OrdersSoapService* for the Service name.
4. Enter a Package as *business*.
5. Enter a Class of *OrdersSoapService* (this is the name of your web service class).
6. Click the Finish button.
7. Inspect the annotations and code generated.
8. Right click on the project and select the New->Other->Web Services->Web Service menu options. Click the Next button.
9. Select the ‘Bottom up Java bean Web Service’ type from the dropdown list. Browse and select *OrdersSoapService* class from the *business* package. Uncheck the ‘Publish the Web service’ and ‘Monitor the Web service’ options. Click the Next button.
10. Check the ‘Generate WSDL file’ and uncheck the ‘Update the default Web.xml’ options. Click the Finish button. This will create a WSDL file located in the *wsdl* folder of your project.
11. Right click on the assignmentEAR project that was created and select the Delete menu option, check the ‘Delete project contents from disk’ option, and click the OK button.
12. Remove the project *assignment* from the server. Click the Finish button. Add the project *assignment* from the server. Click the Finish button. The Add and remove ability can be invoked by right clicking on the JBoss server and selecting the ‘Add and Remove…’ menu. This will clean up the server.

Test your Orders SOAP Web Service calls using the Web Service Tester. To view the Web Service Tester select the Window->Show View->Other->JBoss Tools Web Services->Web Service Tester menu options. This will display the Web Service Tester in your IDE view along with your Console and Servers. To test your service use the following steps:

* + Select JAX-WS from the type dropdown at the top right of tool.
  + Click the ‘Get WSDL’ icon next to the type dropdown.
  + Click the *URL….* button and enter the following URL:

<http://localhost:8080/assignment/OrdersSoapService?wsdl> (*assignment* is your project name)

* + Select the *sayHello* operation. Click the OK button.
  + In the Request Body section of the tool change the arg0 data value from ? to your name.
  + Click *Invoke* button next to the ‘Get WSDL’ icon.
  + This should invoke the *sayHello()* method of the Orders Service and display the Hello message on the Console as well as within the Response Body section of the tool. Take a screen shot of the Response Body in the tool.
* SOAP based Web Service v2:
  + Add new Web Service methods:
    - Open the *OrdersSoapService* class.
    - Inject (use the @Inject annotation) the class scoped variable *service* of type OrdersBusinessInterface.
    - Add a public method named *getOrders()* that returns a List<Order>.
    - Add the @WebMethod() annotations to the *getOrders()* method.
    - In the method implementation call the *service* variables getOrders() method and return the *service* method call as a return value for the method implementation.
    - Regenerate the WSDL file. Right click on the project and select the New->Other->Web Services->Web Service menu options. Click the Next button.
    - Select the ‘Bottom up Java bean Web Service’ type from the dropdown list. Browse and select *OrdersSoapService* class from the *business* package. Uncheck the ‘Publish the Web service’ and ‘Monitor the Web service’ options. Click the Next button.
    - Check the ‘Generate WSDL file’ and uncheck the ‘Update the default Web.xml’ options. Click the Finish button. This will create a WSDL file located in the *wsdl* folder of your project.
    - Right click on the assignmentEAR project that was created and select the Delete menu option, check the ‘Delete project contents from disk’ option, and click the OK button.
    - Remove the project *assignment* from the server. Click the Finish button. Add the project *assignment* from the server. Click the Finish button. The Add and remove ability can be invoked by right clicking on the JBoss server and selecting the ‘Add and Remove…’ menu. This will clean up the server.

Test your Orders SOAP Web Service calls using the Web Service Tester. To view the Web Service Tester select the Window->Show View->Other->JBoss Tools Web Services->Web Service Tester menu options. This will display the Web Service Tester in your IDE view along with your Console and Servers. To test your service use the following steps:

* + Select JAX-WS from the type dropdown at the top right of tool.
  + Click the ‘Get WSDL’ icon next to the type dropdown.
  + Click the *URL….* button and enter the following URL:

<http://localhost:8080/assignment/OrdersSoapService?wsdl> (*assignment* is your project name)

* + Select the *getOrders* operation. Click the OK button.
  + Click *Invoke* button next to the ‘Get WSDL’ icon.
  + This should invoke the *getOrders()* method of the Orders Service and display the Orders as XML within the Response Body section of the tool. Take a screen shot of the Response Body in the tool.

**Short Assignment 5c: Design and build a Message Based Service**

Objective:

1. Design and develop a Message Based Service using a Message Driven Bean.

Activity Directions:

Study the Web Services example code from the reading for this Topic. This activity will add code to the previous activity.

The following steps can be used to setup and configure messaging for use by a Java application.

* Configure Messaging (ActiveMQ) in JBoss:
  + Go the Servers tab within your Project Workspace.
  + Expand the JBoss EAP Server and display the Filesets tree.
  + Expand the Configuration File tree, right click on the standalone.xml file, and select the Edit File menu option. Select the Source tab in the editor.
  + Locate the <extensions> tag.
  + Add the following extension to the bottom of the extensions list:

<extension module="org.wildfly.extension.messaging-activemq”/>

* + Locate the <profile> tag.
  + Add the following configuration within the profile tag (note that a queue ‘Order’ has been added to the configuration):

<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">

            <server name="default">

                <security-setting name="#">

                    <role name="guest" delete-non-durable-queue="true" create-non-durable-queue="true" consume="true" send="true"/>

                </security-setting>

                <address-setting name="#" message-counter-history-day-limit="10" page-size-bytes="2097152" max-size-bytes="10485760" expiry-address="jms.queue.ExpiryQueue" dead-letter-address="jms.queue.DLQ"/>

                <http-connector name="http-connector" endpoint="http-acceptor" socket-binding="http"/>

                <http-connector name="http-connector-throughput" endpoint="http-acceptor-throughput" socket-binding="http">

                    <param name="batch-delay" value="50"/>

                </http-connector>

                <in-vm-connector name="in-vm" server-id="0"/>

                <http-acceptor name="http-acceptor" http-listener="default"/>

                <http-acceptor name="http-acceptor-throughput" http-listener="default">

                    <param name="batch-delay" value="50"/>

                    <param name="direct-deliver" value="false"/>

                </http-acceptor>

                <in-vm-acceptor name="in-vm" server-id="0"/>

                <jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>

                <jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>

                <jms-queue name="TestQ" entries="java:/jboss/exported/jms/queue/TestQ"/>

                <jms-queue name="Order" entries="java:/jms/queue/Order"/>

                <connection-factory name="InVmConnectionFactory" entries="java:/ConnectionFactory" connectors="in-vm"/>

                <connection-factory name="RemoteConnectionFactory" entries="java:jboss/exported/jms/RemoteConnectionFactory" connectors="http-connector"/>

                <pooled-connection-factory name="activemq-ra" transaction="xa" entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"/>

            </server>

* + Locate the ‘jboss:domain:ejb3’ subsystem tag.
  + Add the following configuration with in the ejb3 domain tag:

         <mdb>

                <resource-adapter-ref resource-adapter-name="${ejb.resource-adapter-name:activemq-ra.rar}"/>

                <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>

         </mdb>

* Restart JBoss. Validate that the Messaging is configured properly and no error messages appear when JBoss was started.

Add and update the following components in the project:

* Message based Web Service:
  + Right click on the business package and select the New->Other->EJB->Message Driven Bean menu options.
  + Enter a Class name of *OrderMessageService* (this is the name of your web service class).
  + Enter a Destination name of *java:/jms/queue/Order* (this is the JNDI name of the Order Queue that was added during the Messaging configuration) and leave the JMS and Queue options selected. This will create a queue named *Order* that you can publish messages to. Click the Next and Finish buttons.
  + Inspect the code that was generated. Note, that any messaging received on the Orders Queue needs to be processed in the *onMessage()* method. The business logic for the message handler will be filled in later.
* Update the Orders Business Service v1:
  + Add a new public method definition *sendOrder(Order order)* that returns void to the *OrdersBusinessServiceInterface*.
  + Add an implementation method *sendOrder(Order order)* to the *OrdersBusinessService* implementation class.
  + Add the following class scoped variables to the *OrdersBusinessService* implementation class (note, the annotations parameters are the JNDI names for the Message Connection Factory and Queue that were setup in the JBoss configuration, these are required to send messages to the Order queue):

@Resource(mappedName="java:/ConnectionFactory")

private ConnectionFactory connectionFactory;

@Resource(mappedName="java:/jms/queue/Order")

private Queue queue;

* + Add the following test code, that sends a text message to the Orders queue, in the *sendOrder()* method for the *OrdersBusinessService* implementation class (inspect the code, and think of design changes that could be made to improve the error and exception handling):

// Send a Message for an Order

try

{

Connection connection = connectionFactory.createConnection();

Session  session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

MessageProducer messageProducer = session.createProducer(queue);

TextMessage message1 = session.createTextMessage();

Message1.setText("This is test message");

messageProducer.send(message1);

connection.close();

}

catch (JMSException e)

{

e.printStackTrace();

}

* Update the Orders Message Service v1:
  + Update the *onMessage()* method of the *OrderMessageService* to print the test message to the console (you will have to use a conditional instanceof operator and also typecast the Message parameter to a TextMessage when accessing and use the getText() method).
* Update JSF Response Page and Form Controller:
  + Update the *User* model class from ViewScoped to SessionScoped.
  + Create a new JSF page named *OrderResponse.xhtml* that prints a message ‘[Name], your order was processed!’ with a JSF button (use the h:button tag) that navigates the user back to the TestResponse page.
  + Update the FormController by adding a new handler public method named *onSendOrder()* that returns a string, which calls the *sendOrder()* method on the service class variable and then returns to the Order Response View (i.e. returns a string “OrderResponse.xhtml”). A “test” Order can be created with test data and passed in as an argument to the *sendOrder()* method.
  + Update the *TestResponse* page by adding a JSF command button above the Login Out button with a value of ‘Send Order’ that is processed by the FormController *onSendOrder()* method.
  + Deploy the code and execute the application specifying the TestForm.jsf using the internal IDE browser. Submit the form with valid data. Validate that the test string from the *OrderMessageService* is printed on the Console.
* Update the Orders Business Service v2:
  + Create a new *ObjectMessage* from the session variable and store this in a variable named *message2* of type ObjectMessage.
  + Call the *setObject()* method on the *message2* variable specifying the order variable for the object argument.
  + Send *message2* using the messageProducer variable.
* Update the Orders Message Service v2:
  + Add a class scoped variable named *service* of type OrdersDataService. Add the @EJB annotation above the variable declaration (make sure to import the javax.ejb.EJB packages).
  + Update the *onMessage()* method of the *OrderMessageService* to save the Order to the database (you will have to use a conditional instanceof operator, typecast the Message parameter to a ObjectMessage when accessing, and use the getObject() method to get the MyOrder object, and finally typecast the return value from the getObject () to MyOrder).

Deploy the code and execute the application specifying the TestForm.jsf using the internal IDE browser. Submit the form with valid data. Click the ‘Send Order’ button. Navigate back to the Test Response page to view the Orders. Take a screenshot. The Order created in the Form Controller *onSendOrder()* handler method call should be displayed in the Test Response page.

**Benchmark - Short Assignment 5d: RESTful Web Service (Benchmark and CWF)**

This assignment benchmarks: 1.3 Use object-oriented programming (OOP) to develop client-server and cloud-based applications.

General Requirements:

Use the following information to ensure successful completion of the assignment:

1. This assignment uses a rubric. Please review the rubric prior to beginning the assignment to become familiar with the expectations for successful completion.

Objective:

Create a RESTful web service  
  
Activity Directions:

This is the last project in this course and it includes an individually assessed component, in addition to the team one. As a team, create a set of web services that complement each other. The set of services will provide the following information about a specific section of the Bible, of your choosing. Each team member will choose which of the following services to implement:

* Given a word, the service will return the exact first occurrence of the word (if it exists). The result will include the book name, chapter number, and verse number
* Given a word, the service will return the number of occurrences of that word
* Given a book name, chapter number, and verse number, the service will return the full verse
* Any other service similar in scope and complexity (seek instructor approval)

Each team member will:

1. Create a service class
2. Create a web service using JAX-RS Web Service
3. Deploy the service on a local server
4. Test the service locally
5. Create a Java project as web service client for the web service created by another team member

Deliverables and Submission (**each individual team member**):

1. A document explaining the approach to design and implementation of the service, including a detailed list of all classes, methods, variables, and beans, as appropriate and relevant to your web service (upload to LoudCloud).
2. All Java and HTML source code used to implement, deploy, and test the project uploaded to your GitHub or Bitbucket account (as directed by the instructor).
3. The source code must consist of the 5 components listed above
4. Screenshots demonstrating successful functioning of the service and successful access to a web service created by another team member (upload to LoudCloud).
5. Post the link to your repository on LoudCloud as part of the assignment submission (Bitbucket).
6. An explanation with 150-250 words on how this service might be used to analyze biblical text, find information relevant to current events/issues, and “mine” the biblical text for information not immediately observable during casual reading. How might the service be used if implemented across the entire biblical texts? Provide examples to substantiate your answers (e.g., diagrams, tables).

Deliverables and Submission (**the entire team**):

Important note: These deliverables will be the same for the entire team but anything uploaded into LoudCloud ***MUST*** be uploaded by every team member with his or her individual submission.

1. A document describing the set of services as one complex API for a Bible-based web service. Explaining the approach to design and implementation, including a detailed list of all classes, methods, variables, and beans (upload to LoudCloud).
2. All Java and HTML source code necessary to package all the individually created services as one complex web service (Bible API), uploaded to your Bitbucket account (as directed by the instructor).
3. The source code must consist of the 5 components listed above, but applied to the integrated service.
4. Screenshots demonstrating successful use of the web service and all its components (upload to LoudCloud).
5. Post the link to your repository on LoudCloud as part of the assignment submission (GitHub or Bitbucket).

This assignment uses a rubric. Please review the rubric prior to beginning the assignment to become familiar with the expectations for successful completion.

Developer Notes & Best Practices:

* If you are securing the pages in your web application make sure to configure this properly so that any URI’s for your REST or SOAP based services do not get inadvertently caught in your security framework for your web application.
* You should analyze the complete business and functional requirements before deciding on whether to design a REST or SOAP based service. There are pro’s and con’s to each technology.
* Unless warranted you should never deploy an anonymous REST or SOAP based service especially if the service is exposed over the Internet.
* Unless warranted you should always deploy REST or SOAP based service to utilize secure protocols such as HTTPS.
* You should use the Data Transfer Object (DTO) design pattern as a transport class to return response from a REST or SOAP based service.
* You should be aware of the size and number of entities returned from a service. If necessary, clip the number of entities returned from a service and inform the client thru an error code in the DTO that a partial list of entities was returned.
* Be aware of security policies when returning Personal Identifying Information (PII) data or Payment Card Industry (PCI) data from a service.

Deliverables and Submission:

1. A Project report containing the following:
   * Cover sheet with name of class, assignment, date, and your name.
   * BitBucket URL.
   * Brief theory of operation explaining the approach to design and implementation, including a detailed list of all classes, methods, variables, and pages.
   * Project report with all screenshots showing execution of applications.
2. Upload code to BitBucket
3. Upload code as a zip file to LoudCloud
4. Upload Project report to LoudCloud.

This assignment utilizes a rubric. Please review the rubric prior to beginning the assignment to become familiar with the expectations for successful completion.