Extend Sales Cloud with Oracle PaaS

How to Use, Extend and Deploy UI Accelerator sample application to Oracle PaaS/JCS

Contents

[Disclaimer 4](#_Toc388363558)

[Overview 5](#_Toc388363559)

[Assumptions 5](#_Toc388363560)

[Environment 5](#_Toc388363561)

[Architecture 6](#_Toc388363562)

[What has been developed 7](#_Toc388363563)

[Landing page (Home) – displayed as a separated browser tab (external link in OSC) 7](#_Toc388363564)

[Leads, Opportunities, Customer and Contact pages - displayed as a separated browser tab (external link in OSC) 7](#_Toc388363565)

[Custom page – displayed as a separated browser tab (external link in OSC) 8](#_Toc388363566)

[Custom page – displayed as a Subtab in Oracle Sales Cloud (UI Mashup) 9](#_Toc388363567)

[JWT Token validation message 10](#_Toc388363568)

[Application Setup 11](#_Toc388363569)

[JDeveloper Application 11](#_Toc388363570)

[Shared Library 12](#_Toc388363571)

[Property Files 13](#_Toc388363572)

[Web.xml 13](#_Toc388363573)

[Packaging and Deploying UIAccelerator to Embedded WebLogic Server (On-premise environment) 15](#_Toc388363574)

[Packaging and Deploying UIAccelerator to Oracle Java Cloud Service 18](#_Toc388363575)

[Oracle Sales Cloud Integration setup – Application Composer 24](#_Toc388363576)

[Application Design 27](#_Toc388363577)

[fusionSubtab.jspx 27](#_Toc388363578)

[sampleOptyTF 28](#_Toc388363579)

[viewOpportunities.jsff 29](#_Toc388363580)

[Extending the UI Accelerator sample code 30](#_Toc388363581)

[What are you building? 30](#_Toc388363582)

[Generating ADF Web Service Proxy Project 30](#_Toc388363583)

[Using JWT User Token API for Web Service Proxy 37](#_Toc388363584)

[Creating a Bean Data Control to expose Web Service operations 48](#_Toc388363585)

[Building the UI 50](#_Toc388363586)

[Testing the Application 56](#_Toc388363587)

# Disclaimer

All sample code is provided by Oracle for illustrative purposes only.

These sample code examples have not been thoroughly tested under all conditions. Oracle, therefore, cannot guarantee or imply security, reliability, serviceability, or function of the sample code.

All sample code contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

This sample code illustrates the use of JWT UserToken for invoking Fusion applications web services with end user context. Note that providing Web SSO experience solely based on successful validation of Fusion Cloud User Assertion (JWT User Token) in the URL parameter of the iFrame request is highly discouraged since it has the effect of elevating the exposure resulting from any breach of user’s browser session by malicious “man in the middle”.

# Overview

The purpose of this document is to guide you how to use, extend and deploy the UI Accelerator sample application. This application is a Fusion Web Application (ADF application) developed using the full ADF stack, such as ADF Rich Faces Components (i.e. DVT/tables/skinning), ADF Controller (Task Flows) and ADF Business Components (Data Controls) to enable your application to easily integrate with Oracle Sales Cloud.

The fundamental idea of this sample application is to allow developers to build their application rapidly. It should allow them to understand and explore some of the integration patterns available between Oracle Sales Cloud (SaaS) and Oracle Java Cloud Service (PaaS). This sample application leverages a set of code for the view layer, keeping the UI with the same look and feel of Oracle Sales Cloud R8 (Simplified UI). In addition, this application contains an API so that developers, after creating a Java Proxy Client project to consume OSC Web Services, they can make use of that set of classes to setup security/authentication policies easily. Sometimes, that process might be tricky during the development process and using this API can save your time. Finally, there is a task flow which allows the application to validate the JWT User token, avoiding invalid calls to OSC Web Services using an invalid token value.

## Assumptions

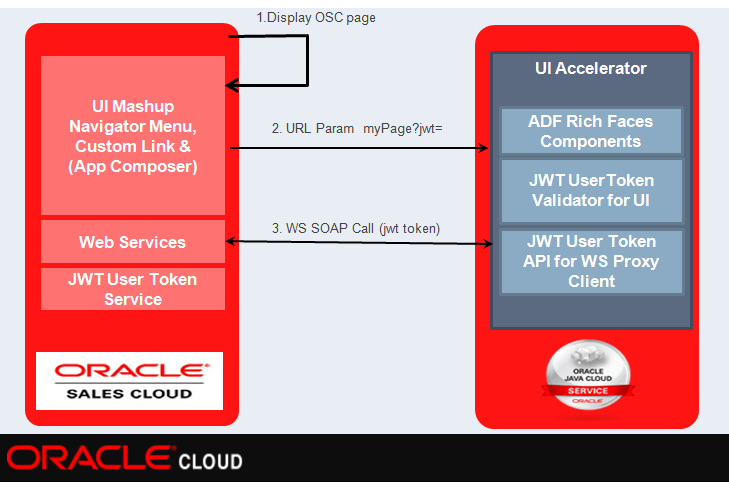
* It is assumed that the reader is competent with Oracle JDeveloper, Oracle ADF and general Java development, as well as Oracle Sales Cloud extensibility concepts (i.e. Application Composer).
* Reader is aware of creating jar, war, ear deployment packages
* It’s also assumed that reader is aware of WebLogic deployment

## Environment

* Oracle Sales Cloud Release 8
* Oracle Java Cloud Service 13.2+
* Oracle JDeveloper 11.1.1.7.1

## Architecture

Every request to the UI Accelerator application starts from either an external link or UI Mashup in Oracle Sales Cloud. Application Composer/Groovy script is used to build that integration, which is responsible for creating an URL to the application “UI Accelerator” passing the JWT user token value as a URL parameter. Then, the application contains a Task Flow which validates the jwt user token value. If the token value is valid and, in addition, there is a requirement to consume Oracle Sales Cloud web services, there is also the possibility of using an API available to help developers to build OSC requests without the need to know in details what are the required parameters used to setup authentication header or web service policies. This API makes the process of setting up web service security policies very straightforward for developers and avoids silly errors, mitigating the effort developed during the process of building an application.



## What has been developed

Below you can find a summary of what type of pages has been created on UIAccelerator sample application and what integration scenario can be exploited with those pages. All those pages have the same look and feel of OSC Simplified UI:

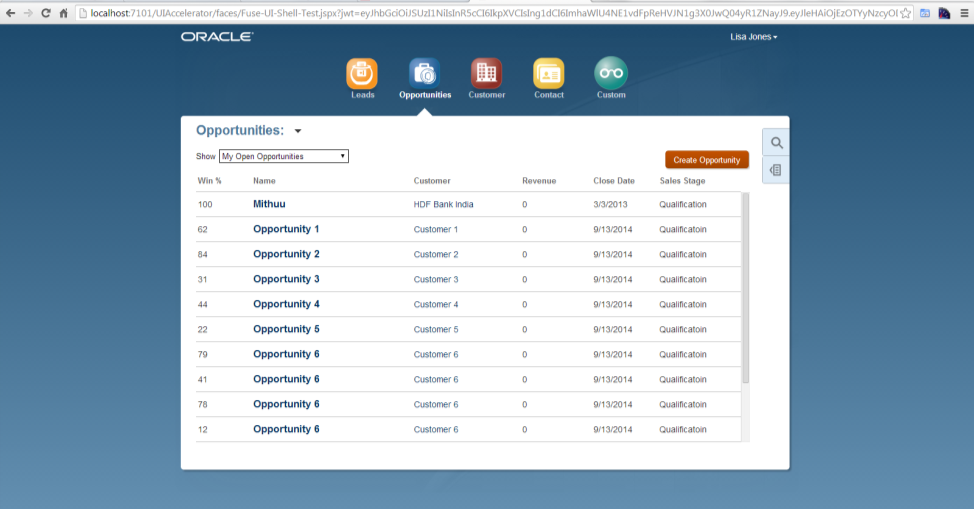
## Landing page (Home) – displayed as a separated browser tab (external link in OSC)

This landing page should be called from an External Link in Oracle Sales Cloud. This integration pattern usually is used when is expected that your application has its own visual identity or at least to identify that user is navigating on an application off the boundaries of OSC.



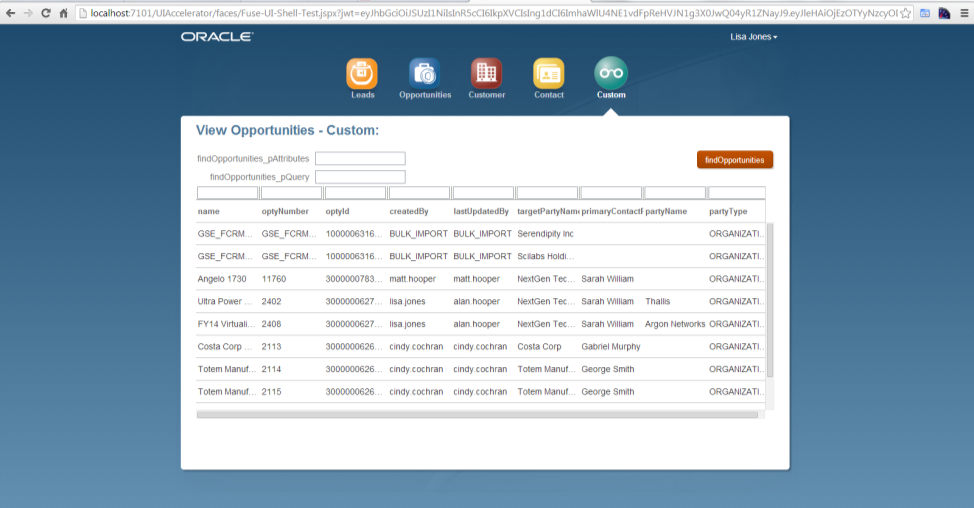
## Leads, Opportunities, Customer and Contact pages - displayed as a separated browser tab (external link in OSC)

These pages are static ones, available when users start navigating from the landing page. They contain hard coded pieces which help developers to understand the best practices of how to create good layouts/designs using ADF and Fuse Skin (Simplified UI look and feel).



## Custom page – displayed as a separated browser tab (external link in OSC)

This is a dynamic page, accessed from the landing page, which uses a Data Control binding operation to consume data from OSC. Behind the scenes, the data control is a wrapper class which uses a Java Proxy Client to consume OSC web services.

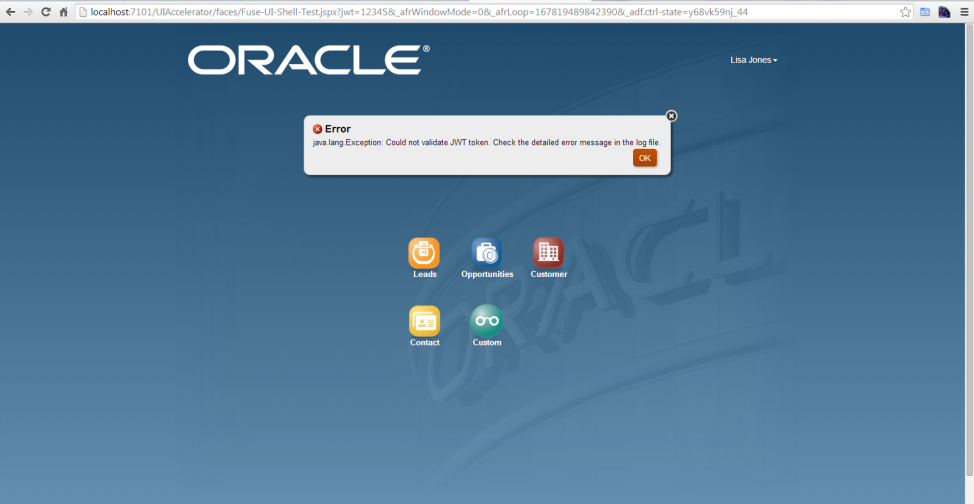


## Custom page – displayed as a Subtab in Oracle Sales Cloud (UI Mashup)

This is the same page as above (dynamic one), however, it’s being re-used for another integration scenario, where the page can be accessed from a Subtab (UI Mashup) of an Object page in Oracle Sales Cloud. At the picture below, you can see that from the Edit Household page in OSC, there is an icon (thumbs up) which displays that page as an iframe.

## JWT Token validation message

Every time users trying to consume a web service in Oracle Sales Cloud using an invalid user token value, the application will deny the access and an error message should be displayed. There is also a possibility to validate the jwt user token by using the jwt token validation task flow. This will validate the jwt token value passed as URL parameter.



# Application Setup

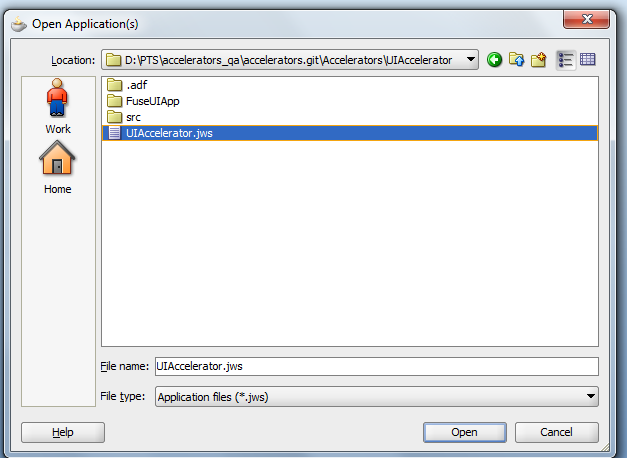
Now that you are aware of the big picture of the application, you can start exploring each piece of the application individually.

## JDeveloper Application

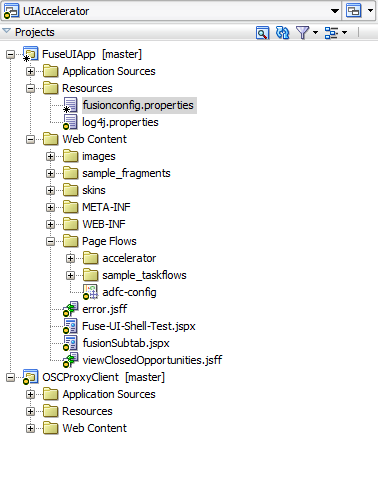
First step is to open your Application in JDeveloper. Our assumption is that you either have extracted the zip file into a folder in your machine or pulled out the source code from a source control repository (git, subversion, etc).

Start JDeveloper 11.1.1.7.1 and click on Application -> Open

Locate the folder where all the source code is located (Accelerators\UIAccelerator). Select UIAccelerator.jws file and click on Open.



Then the following project structure should show up in your JDeveloper:



Basically the main application is composed by one project (FuseUIApp) which contains three main folders and a set of Data Controls:

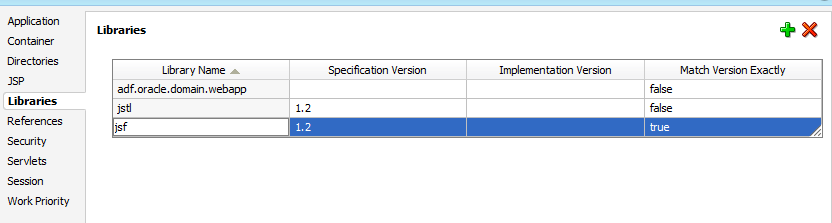
* Application Sources 🡪 application source code (managed bean, utils, etc), xml binding files, etc.
* Resources 🡪 ADF Library dependency folder, property files related to Oracle Sales Cloud connection settings and log messages.
* Web Content 🡪 ADF Task Flows, JSPX pages, JSFF files (page fragments), deployment descriptor files (web.xml, weblogic.xml, etc), style classes, images, libraries, etc.
* Data Controls 🡪 Some of Oracle Sales Cloud Web Services were exposed as a Data Control on RestFul accelerator asset. Some business operations are available for a specific set of web services (Find opportunity, Find Person, etc).

There is also another project called OSCProxyClient which is the integration project used to consume Oracle Sales Cloud web services.

## Shared Library

Since JCS 13.2, it’s possible to deploy and re-use ADF shared libraries available in Oracle Java Cloud Service. On UIAccelerator application we are referencing adf, jsf and jstl libraries. It’s required to add an entry for each library we want to reuse into the weblogic.xml deployment descriptor file. More information about JCS shared-libraries is available on [Guidelines for ADF Applications](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop002.htm#BCEFFHEH) documentation. Check below the content of the weblogic.xml file:

WEB-INF\weblogic.xml

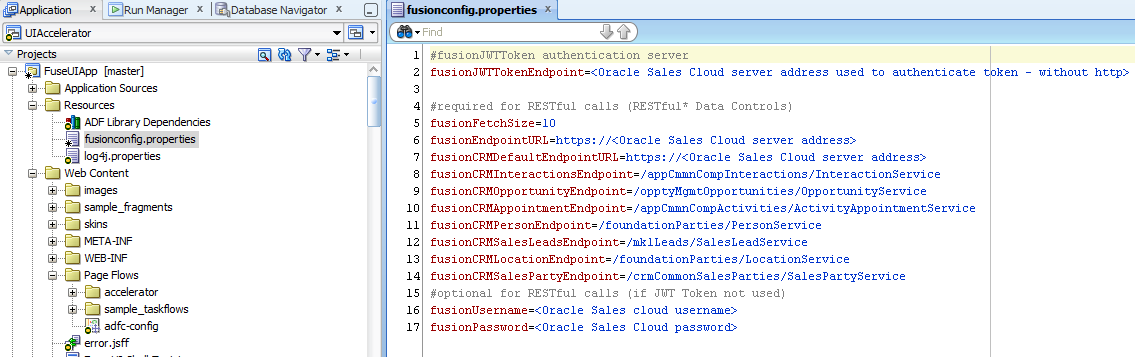


## Property Files

Before deploying the application is required to update the properties used to connect to Oracle Sales Cloud web services, i.e., service endpoint URL, username and password properties. There are two distinct service endpoints, one is used exclusively to authenticate user against JWT UserService web service (fusionJWTTokenEndpoint property) and another one is used for general OSC Web Services access when you have a Java Web Service Proxy Client (fusionEndpointURL property).

Note: If you are going to use JWT token as the authentication method, you are not required to setup username and password properties.

The properties file is used here for convenience only. We recommend that, for production deployment, these properties should be stored in a database table in JCS/DBCS.



## Web.xml

There are some additional settings which might be required when you deploy your application to JCS and want to call it from within an UI Mashup (iframe) in Oracle Sales Cloud. One case is the need to set FRAME\_BUSTING = never. There are other setting which can be used that affects ADF Application performance as described below:

<!-- context-param related to performance setup and UI subtab in OSC-->

<context-param>

<description>If this parameter is 'always' than The page will show an error and redirect whenever it attempts to run in a frame. If this parameter is 'differentOrigin'(default value) than The page will show an error and redirect only when it attempts to run in a frame on a page that originates in a different domain . If this param is 'never' than The page can run in any frame on any originating domain.</description>

<param-name>org.apache.myfaces.trinidad.security.FRAME\_BUSTING</param-name>

<param-value>never</param-value>

</context-param>

<context-param>

<description>#Performance setting</description>

<param-name>org.apache.myfaces.trinidad.DISABLE\_CONTENT\_COMPRESSION</param-name>

<param-value>false</param-value>

</context-param>

<context-param>

<description>#Performance setting</description>

<param-name>oracle.adf.view.faces.CLIENT\_STATE\_MAX\_TOKENS</param-name>

<param-value>3</param-value>

</context-param>

<context-param>

<description>#Performance setting</description>

<param-name>org.apache.myfaces.trinidad.COMPRESS\_VIEW\_STATE</param-name>

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

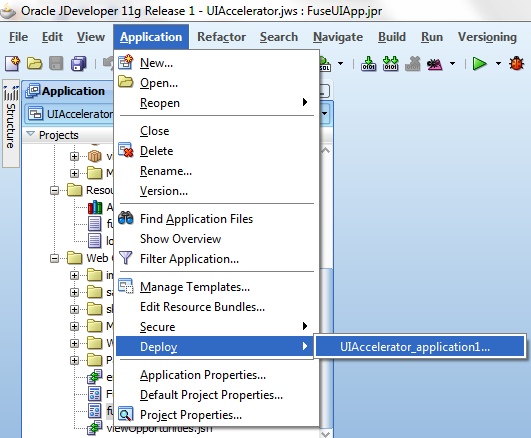
</context-param>

How to Deploy UIAccelerator sample application

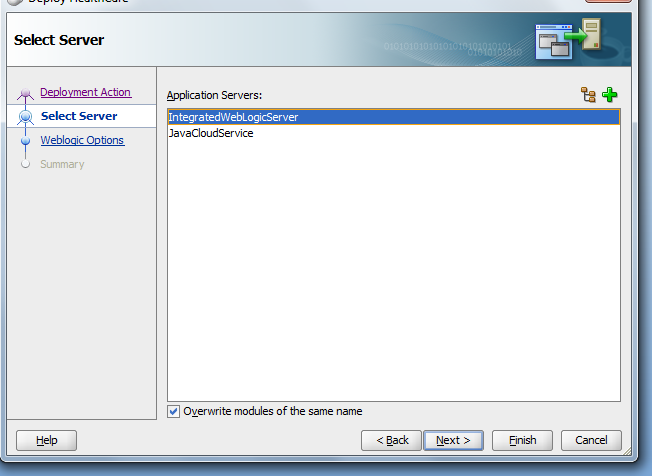
Usually developers do a dry run test on his local WebLogic server before deploying the application to Oracle Java Cloud Service. This is a good practice but there is no guarantee that your application should works on a Cloud environment, because there is some requirements/specification exclusive for applications hosted in Oracle Java Cloud Service. Check the typical Workflow for using the Oracle Java Cloud Service on [JCS documentation](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop.htm#CSJSU7163).

## Packaging and Deploying UIAccelerator to Embedded WebLogic Server (On-premise environment)

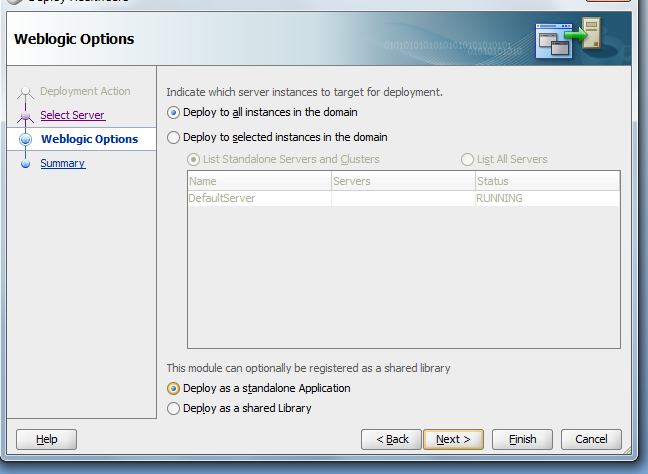
It’s time to package your application and deploy it to embedded WebLogic server. The application was designed to be deployed as an “.ear” deployment package since there are some Data Controls (Business classes) created. In JDeveloper, go to Application 🡪 Deploy 🡪 UIAccelerator\_application1.



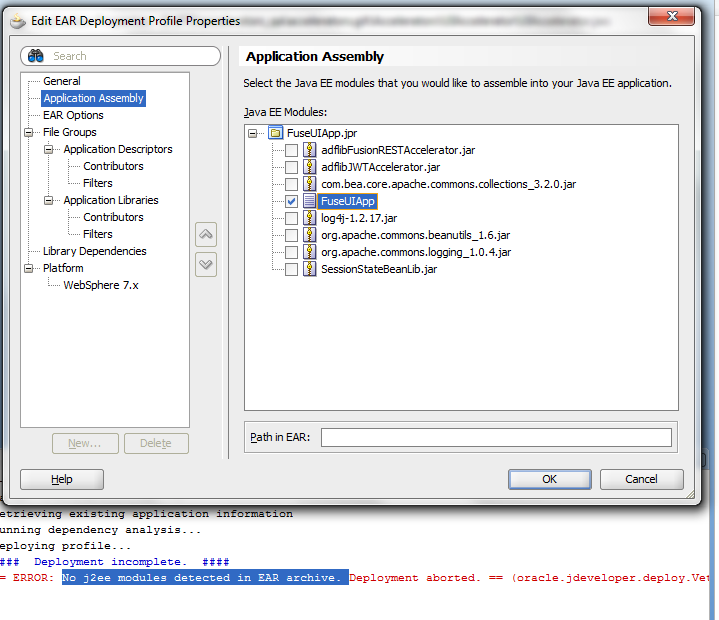
On Deployment Action screen, select Deploy to Application Server, and click on Next. Then, select IntegratedWeblogicServer.



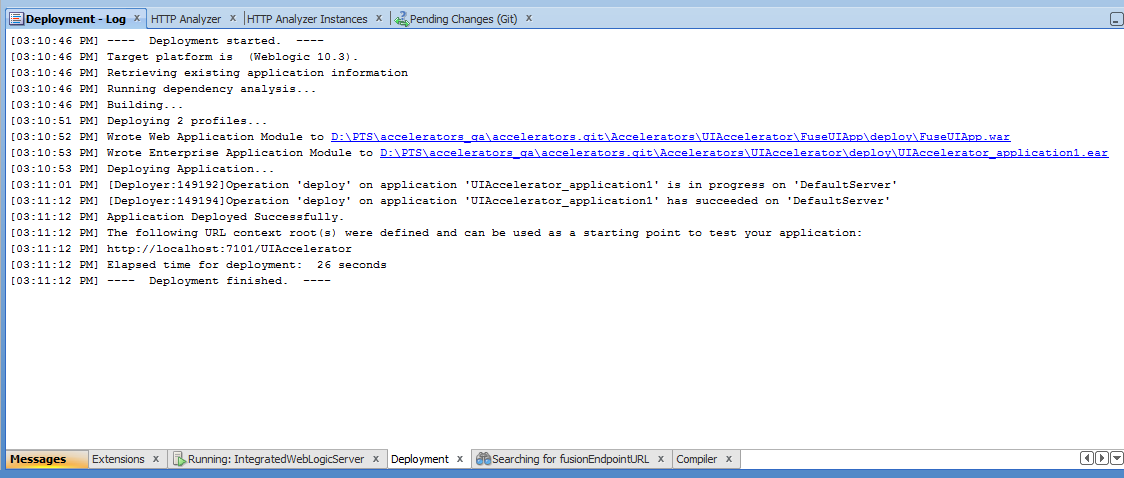
On WebLogic Options screen, use Default options and click on Finish.



It would take a while. Check the status of the deployment in the Deployment – Log tab. If you got an error message stating that there is no J2EE modules detected in the EAR file, go to Application 🡪 Application Properties and double check if FuseUIApp module (WAR) was “checked” under Application Assembly screen.



Otherwise, observe the Deployment – log window:



Note: make sure you are deploying the .EAR file by using the “Application” menu, not “Project” menu.

There are 2 distinct URLs available as the result of the deployment process, which you can use to have access to the application (as explained on “What has been developed” topic above). Assuming the JEE Context root setup in the application is “/UIAccelerator”, the root URL of the application is:

http://localhost:7101/UIAccelerator

If you try to open up that link directly, you should get an error as displayed below:



That happened because there is no index.html page redirecting you to the location that the pages are available or either because the required input parameter is missing.

If you want to display the page which contains the Landing Page, you need to use the following URL:

[http://localhost:7101/UIAccelerator/faces/Fuse-UI-Shell-Test.jspx?jwt=<jwtToken>](http://localhost:7101/UIAccelerator/faces/Fuse-UI-Shell-Test.jspx?jwt=%3cjwtToken%3e)

If you want to display the Custom page directly (from the UI Mashup), you need to use the following URL:

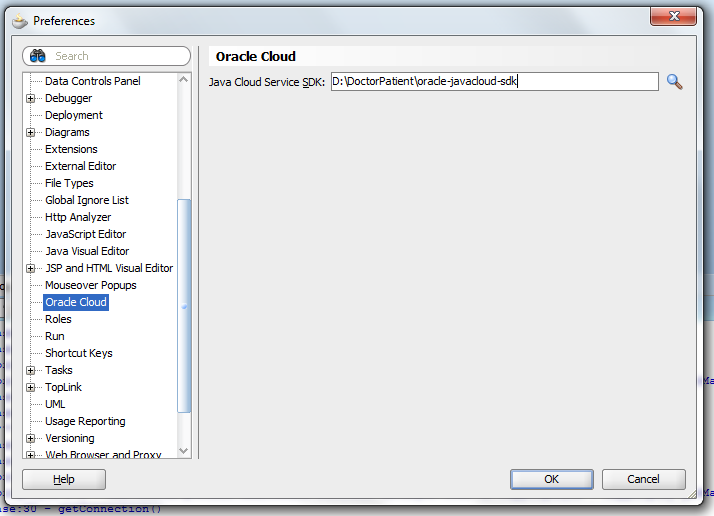
[http://localhost:7101/UIAccelerator/faces/fusionSubTab?jwt=<jwtToken>](http://localhost:7101/UIAccelerator/faces/fusionSubTab?jwt=%3cjwtToken%3e)

Where we have <jwtToken> = OSC JWT Token dynamically created

## Packaging and Deploying UIAccelerator to Oracle Java Cloud Service

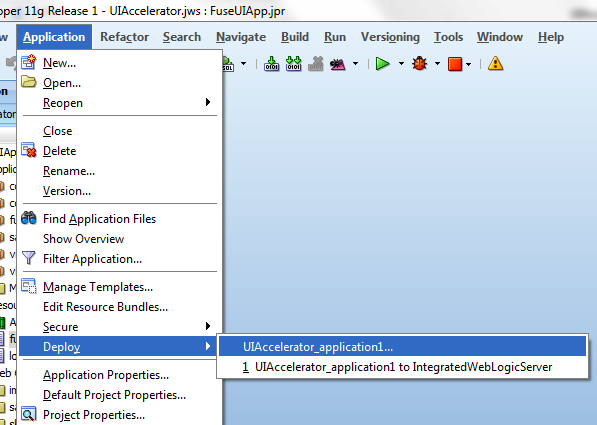
Since you developed and tested the application in your embedded WebLogic server, you are now almost ready to deploy your application to Oracle Java Cloud Service. Before deploying the application, it’s important to set up your JDeveloper to use JCS SDK. More information on how to download and install JCS SDK is available [here](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop001.htm#BCEFGHAF).

For now, after downloading and installing JCS SDK in your local machine, you need to go to JDeveloper preferences and update Oracle Cloud settings with the location of SDK Classpath.

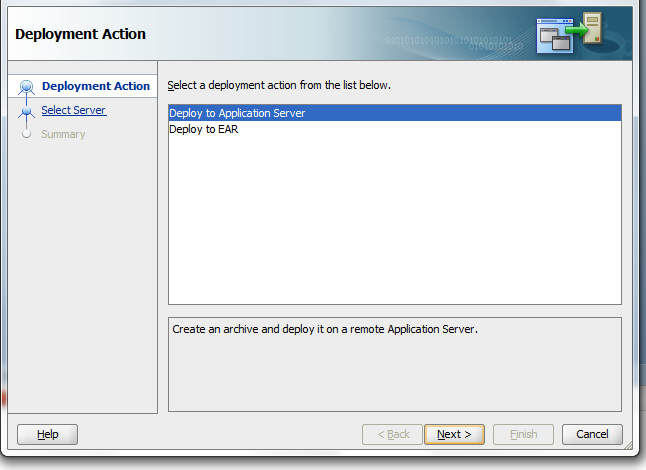


Ok, let’s start working with JCS. Since the application is working on a local environment, we should promote the application to JCS. The same instructions we used to create a connection to embedded WebLogic Server in JDeveloper, should be followed to create a connection to JCS.

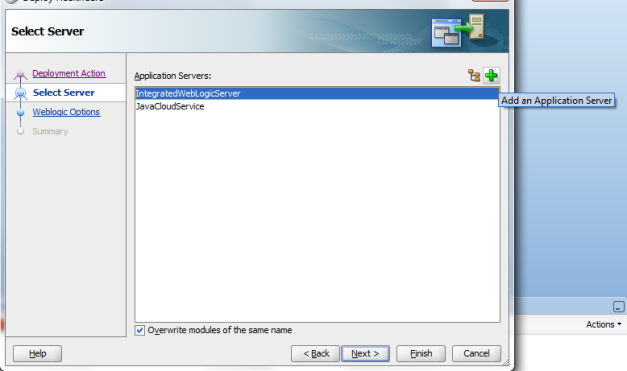
Go to Application 🡪 Deploy 🡪 UIAccelerator\_application1…



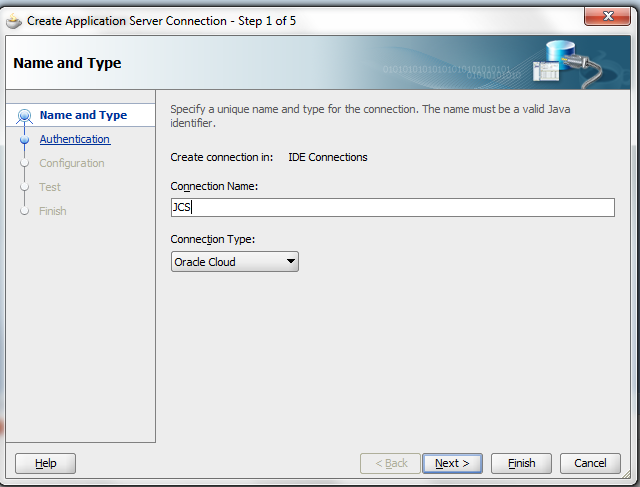
Select Deploy to Application Server:



Click on the plus icon (+) to create a new connection



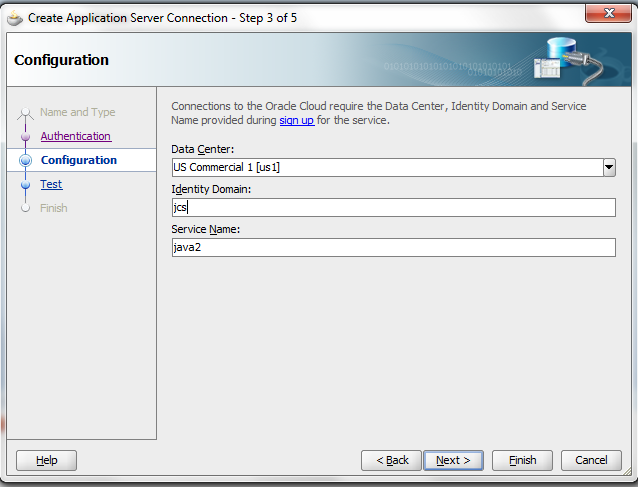
Type in a name for your connection and choose Oracle Cloud as a Connection type.



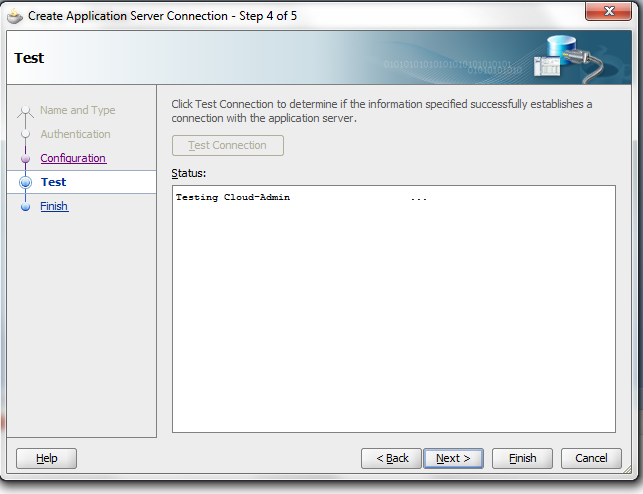
Fill out the form with the Username and Password for your JCS connection:



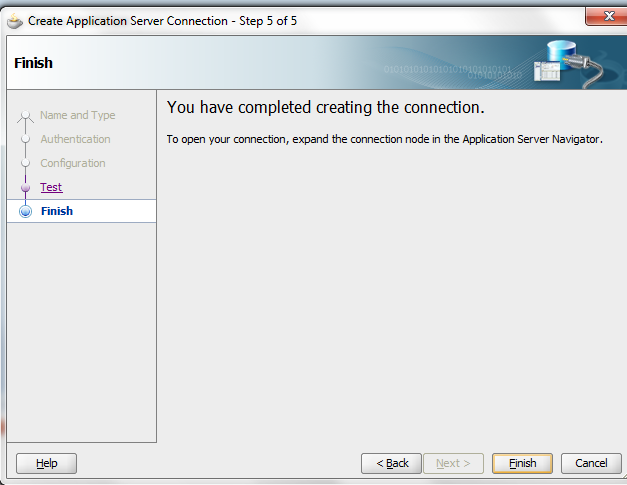
Provide the connection details related to the Data Center, Identity Domain and Service Name:



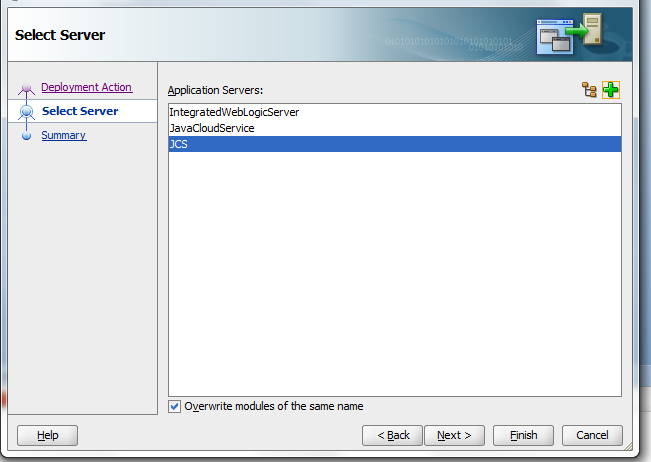
Test your connection:



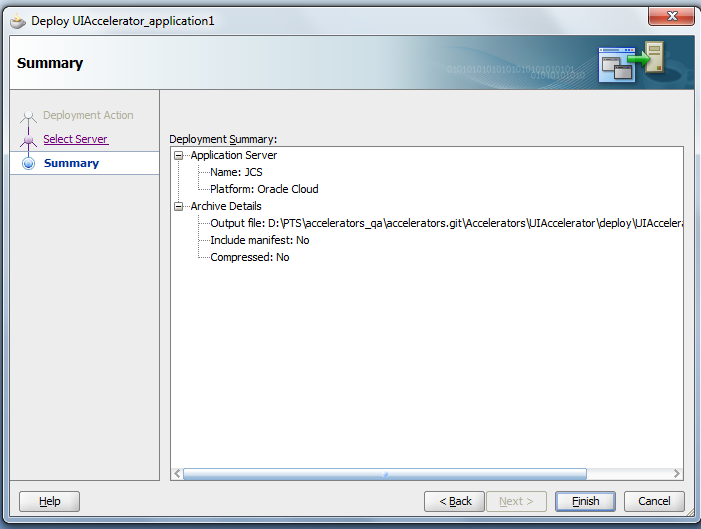
If you successfully connect to JCS, click on Next and then, click on Finish button to complete your configuration to the service.



Now that your connection is ready, you just have to select the Application Server connection you created and click on Next to deploy the package.



Check the Deployment Summary and click on Finish to start deploying the application to Oracle Java Cloud Service.



This would take a while; check the deployment log to follow the progress.

## 

## Oracle Sales Cloud Integration setup – Application Composer

In order to call the pages you hosted in Oracle Java Cloud Service from an object page in Oracle Sales Cloud (i.e. Opportunity, Household, Contact pages, etc), you are required to decide what integration design you will use to display the page: UI Mashups or External Links. After that, you have to start working with Application Composer. The Application Composer is one of the main tools used to extend Oracle Sales Cloud. This can be used to create UI Mashups or External Links, as well as to create Custom Objects or Custom Fields in OSC. More details can be found [here](http://docs.oracle.com/cloud/latest/salescs_gs/salescs_integrate.htm).

In order to generate a link/URL to the UIAccelerator sample application, you have to build up a Groovy Script using the App Composer:

Go to Application Composer 🡪 Application Customer Center 🡪 Sales Account 🡪 Actions and Links 🡪 Create a new Link. Then, add the created link to Edit Summary Form section of Desktop UI.

def serverAddress = "<oracle java cloud service address>"

def jwt = (new oracle.apps.fnd.applcore.common.SecuredTokenBean().getTrustToken())

def finalURL = "https://"+serverAddress+"/UIAccelerator/faces/Fuse-UI-Shell-Test.jspx?jwt=" +jwt

return(finalURL)

Since OSC R8, it may be required to add Partner Endpoint to Topology Manager, instead of hard coding the server address to the groovy script. So that, you will not have to change the groovy code every time that the service endpoint changes value (e.g. moving the application from QA instance to production).

Below, there is a sample scenario to register the application on Topology Manager:

Go to Setup and Maintenance 🡪 Topology Objects 🡪 Manage Third Party Applications

//Setup the FULL URL (including server port)

Application Name: UIAcceleratorApp

Full URL = https://<oracle java cloud service>/UIAccelerator/faces/fusionSubtab

To create a UI Mashup in Opportunity Page for Simplified UI, go to Application Composer 🡪 Sales 🡪 Standard Objects 🡪Opportunity 🡪Pages 🡪Simplified Pages tab 🡪Details Page Layout 🡪Edit Layout. And then create a new subtab, adding the following Groovy script:

//retrieve JWT Token value

def crmkey= (new oracle.apps.fnd.applcore.common.SecuredTokenBean().getTrustToken())

//build URL parameter

def urlParam = "&jwt="+crmkey

//retrieve service endpoint

def serverAddress = oracle.topologyManager.client.deployedInfo.DeployedInfoProvider.getEndPoint("UIAcceleratorApp ")

//return URL

return serverAddress+urlParam;

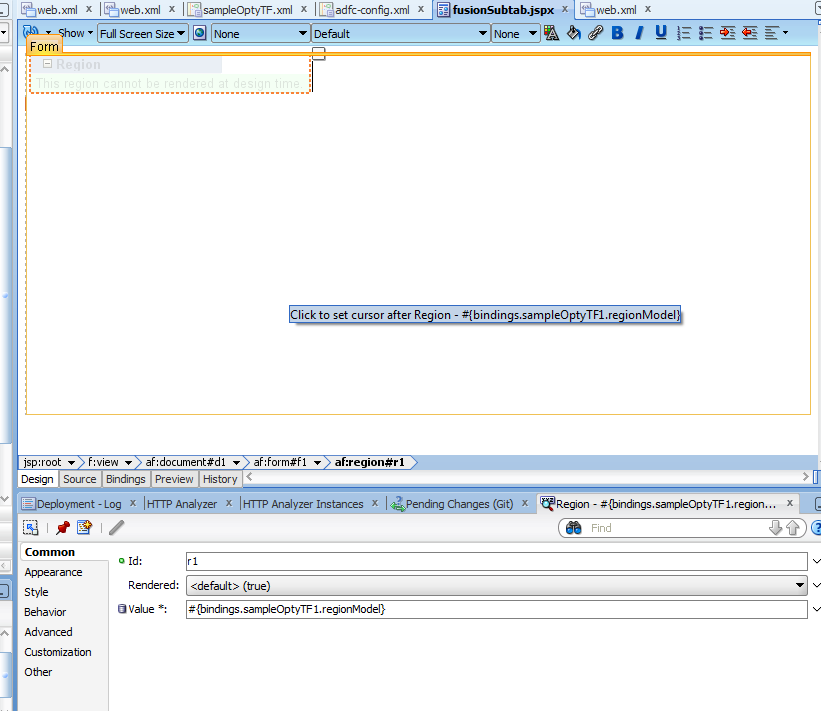
The result of those pieces of code is an URL similar to the one below:

https://<oracle java cloud service address>/UIAccelerator/faces/fusionSubtab? jwt=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6ImhaWlU4NE1vdFpReHVJN1g3X0JwQ04yR1ZNayJ9.eyJleHAiOjEzOTYyNzcyODMyOTQsImlzcyI6Ind3dy5vcmFjbGUuY29tIiwicHJuIjoibWF0dC5ob29wZXIiLCJpYXQiOjEzOTYyNzU0ODMyOTR9.Khh9JerRQfz4kHRkVlotVLwCr6bv6Tbb1z8SSm5jKfcIxmNm890x8inpBJXmGAx9JU5ueSbXz2mZrLxXISX65D2VU6Q0Cnuxx3XMMxNpIon-tRGxmSih3pc6ZguGolkTUvYa5GyByTu6OgIK2MUy2M\_qF1Y8hwT1-h\_BhM-ynAE

# Application Design

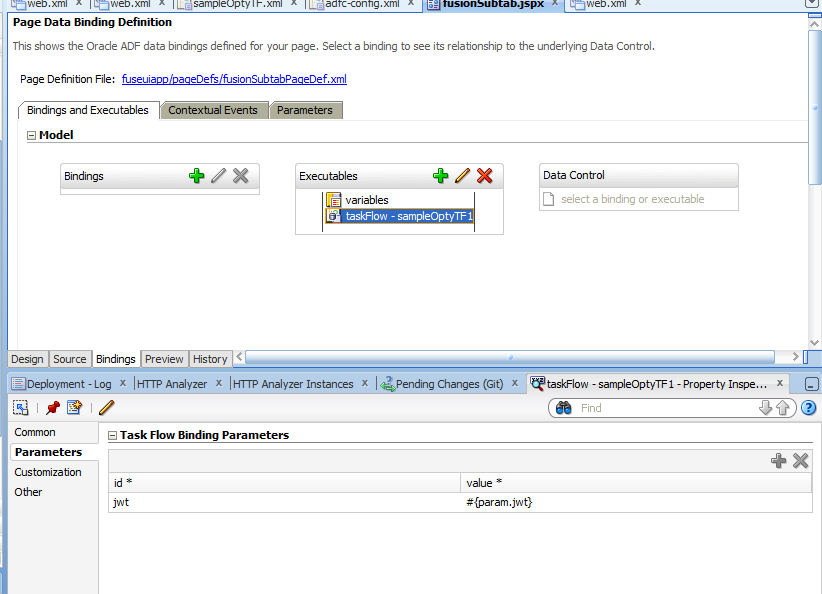
## fusionSubtab.jspx

This page is a container for the sampleOptyTF bounded task flow. There is a static region designed to render that task flow.



Since we are using the JWT User token as the authentication mechanism to invoke OSC Web Services and that token is coming from the OSC UI Mashup as an URL parameter. There is a need to read and parse that information from the URL and then pass it over to the bounded task flow.

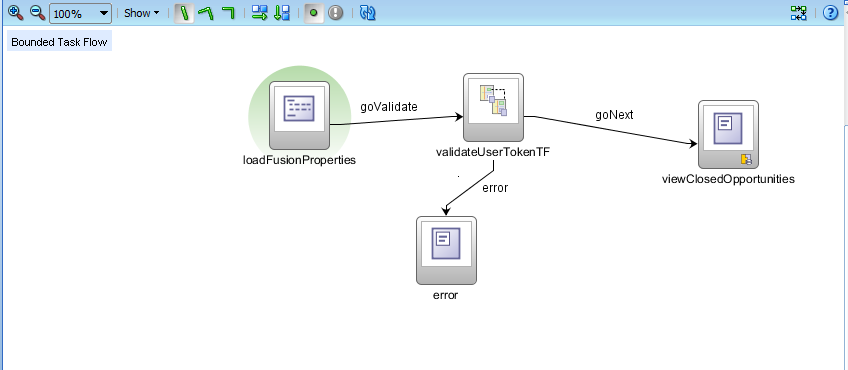
In order to read that URL parameter and assign the value to the task flow binding parameter, you should use the expression language displayed below. If you want to change the binding parameter value, click on Pencil icon:



## sampleOptyTF

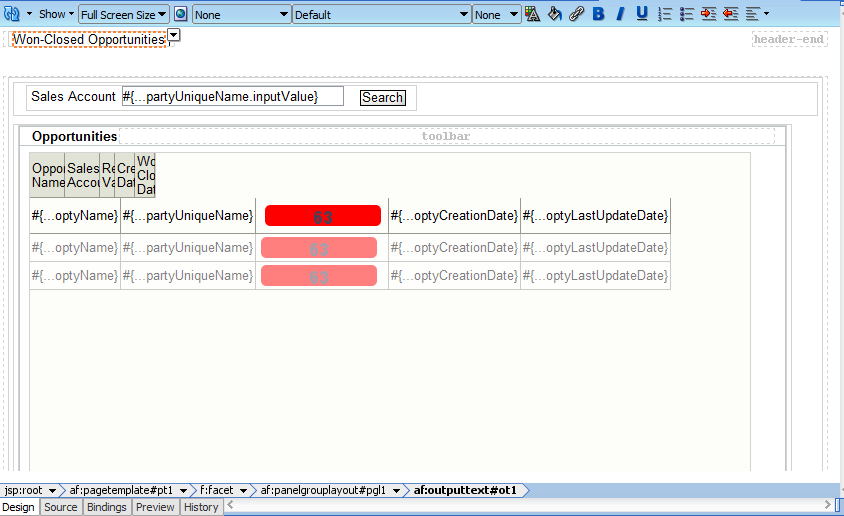
Next step is sampleOptyTF bounded task flow. This one is responsible to read parameters from the property file and save into the pageFlowScope variable, then, call the validateUserTokenTF (JWT UserToken Accelerator) which contains the logic to validate the value of the JWT User token against the OSC, before calling the viewOpportunities view (page).

Only to recap, the JWT User token contains user context information and should replace username/password Basic Authentication to invoke Oracle Sales Cloud Web Services. In case the jwt is invalid by timeout or invalid token value (user trying to bypass security), user should be redirect to the error page.



## viewClosedOpportunities.jsff

Finally, we have the view responsible to render the content by the task flow. This fragment contains a form to query a data in Oracle Sales Cloud and to render a list of Opportunities in a table.



# Extending the UI Accelerator sample code

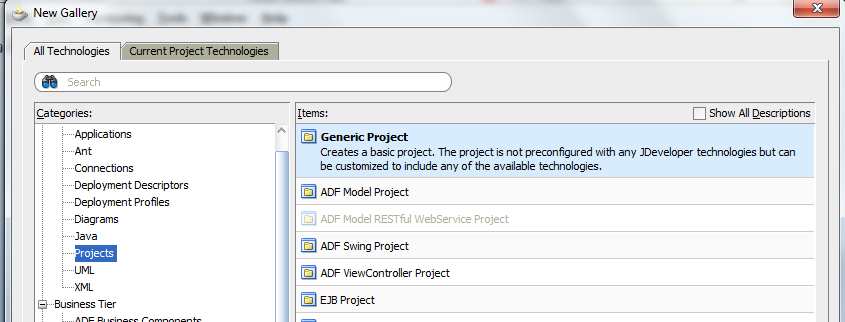
Now that you could understand what are the main components of the application you are able to create a new page, consume Oracle Sales Cloud Web service by using JWT User token and finally display your page as an UI Mashup in OSC, using ADF Rich Faces components.

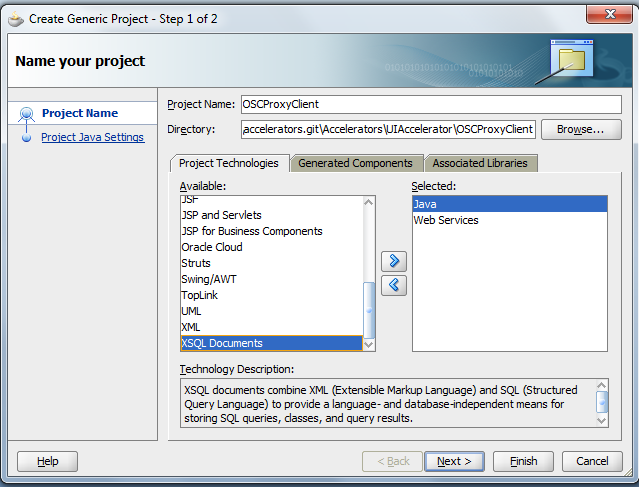
## What are you building?

You are going to build a simple use case which displays a form to search for Closed/Won opportunities. There is a need to generate a Web Service Proxy Client to query data from OSC. The resulting data is presented in a table with ADF dvt:gauge graph component embedded.

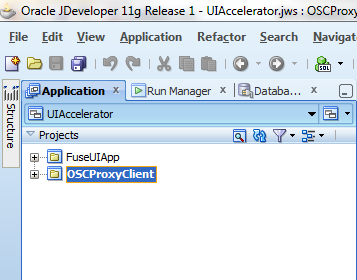
## Generating ADF Web Service Proxy Project

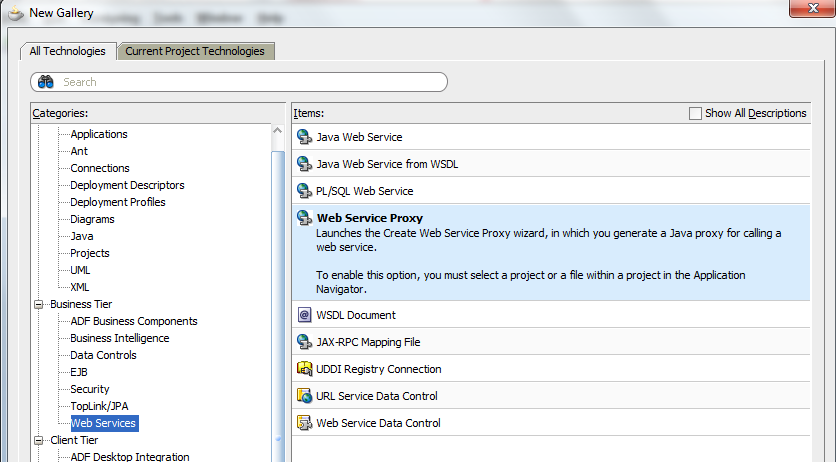
It’s a good practice to create a new Project for each Web Service Proxy Client you required. To sum up, you can find a screenshot displaying how to create a new web service proxy client project for OSC Opportunity Service in JDeveloper:

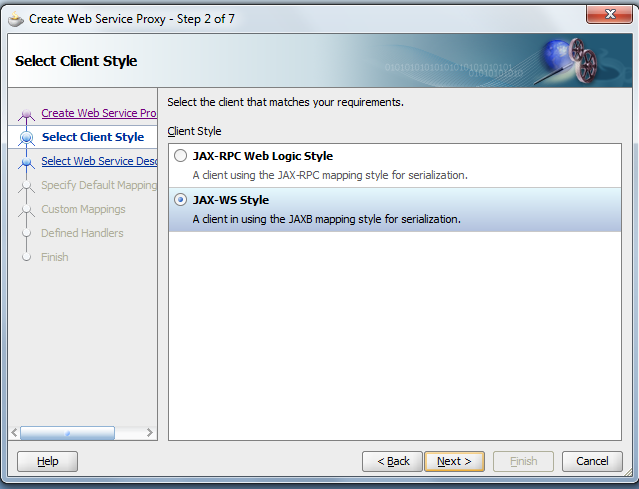






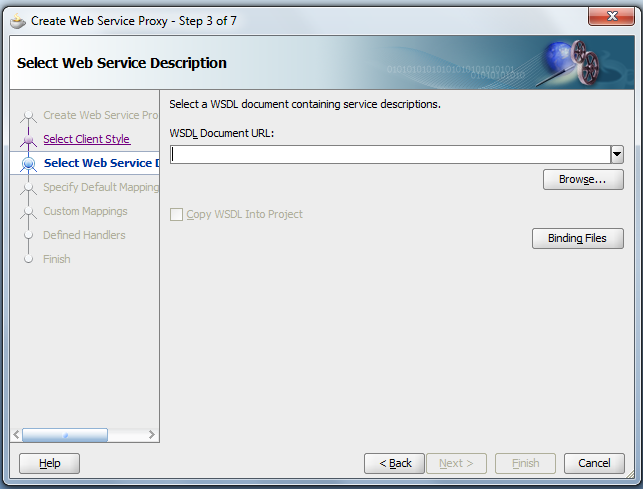


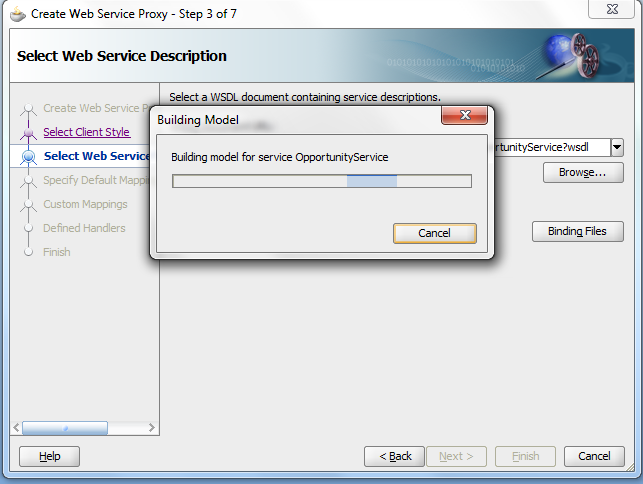


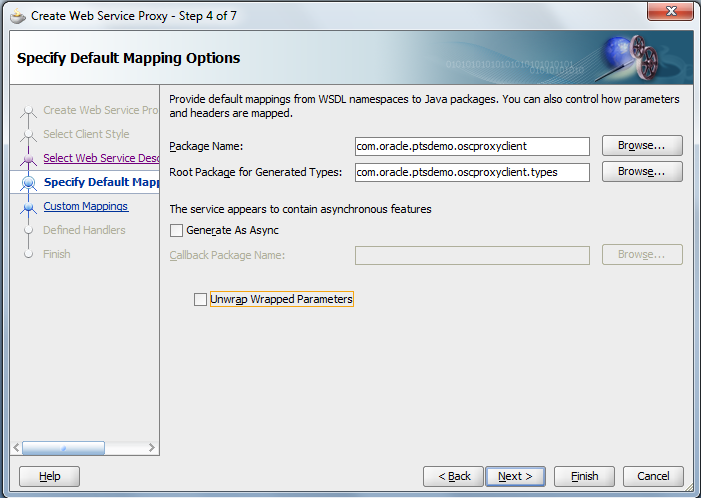


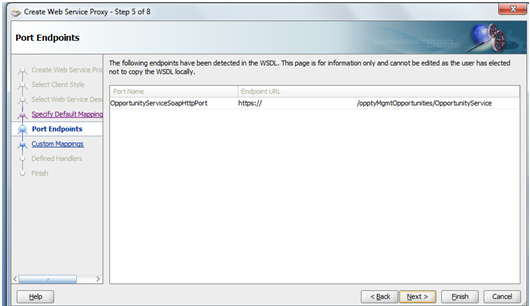
Opportunity Web Service Endpoint:

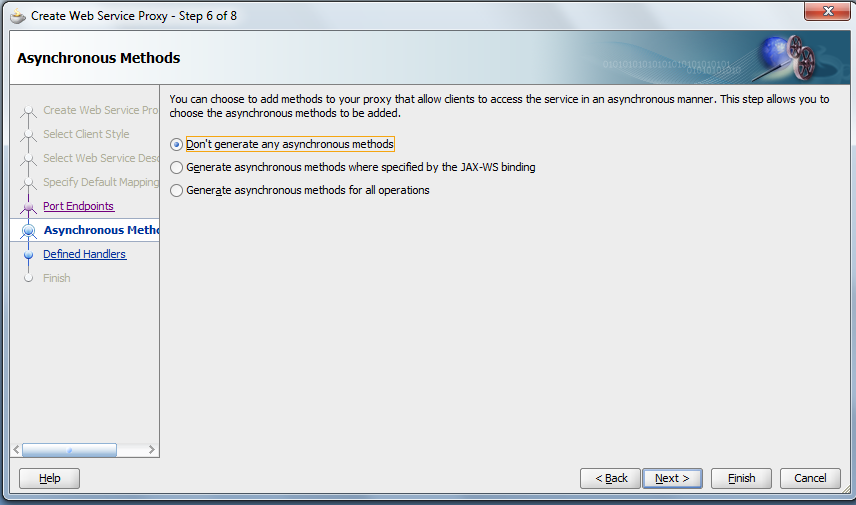
https://<server:port>/opptyMgmtOpportunities/OpportunityService?wsdl

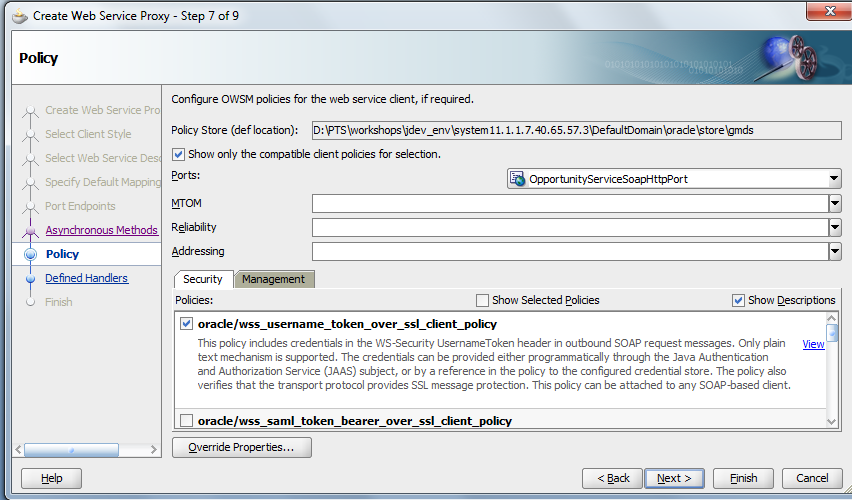


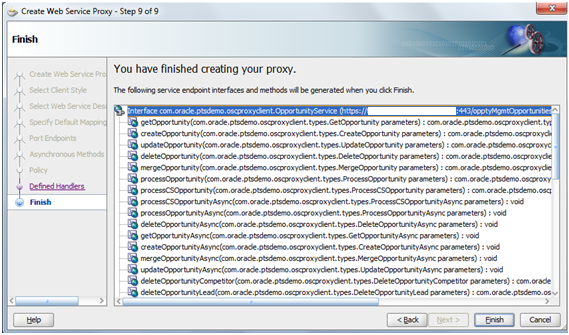


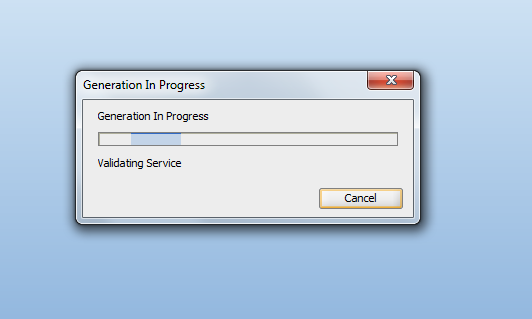




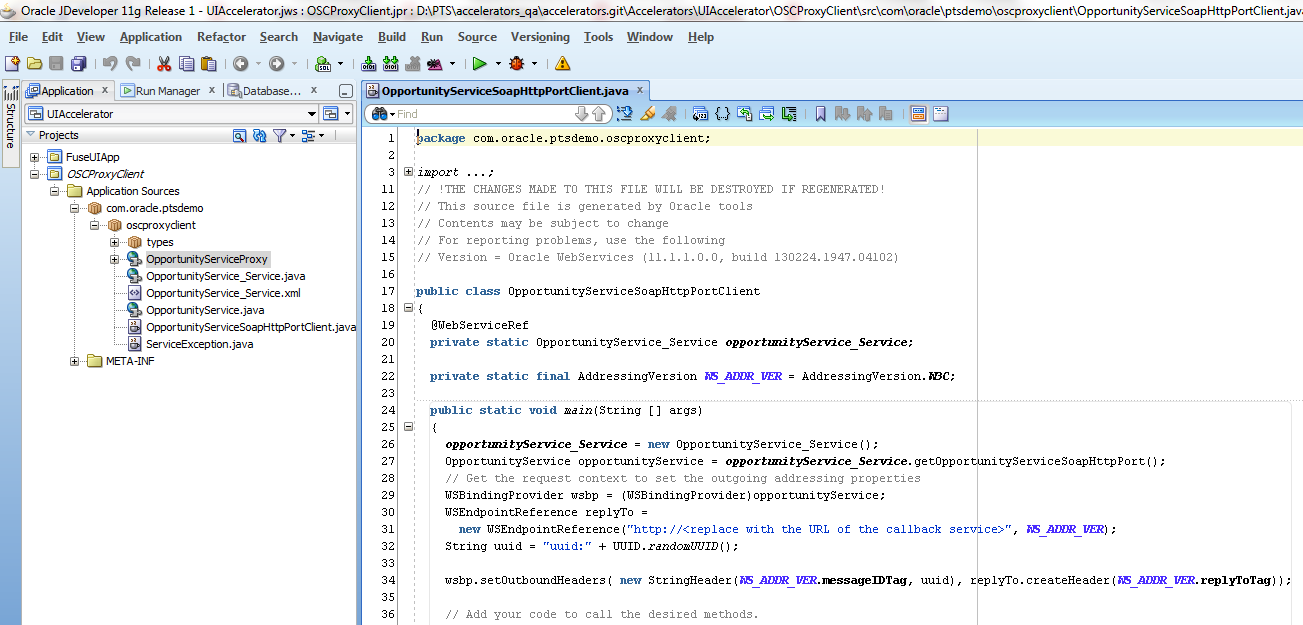






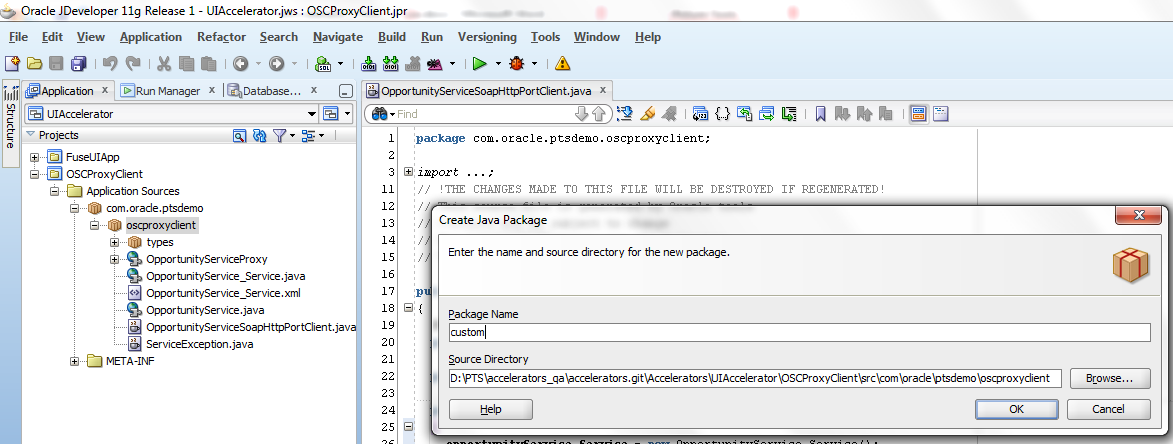


JDeveloper automatically generates a “PortClient” class in your project: e.g. for Opportunity we have “OpportunityServiceSoapHttpPortClient”. For now, you can just ignore that class because we are going to generate a new one, a custom class. You can use that one, however, keep in mind that this class is overwritten whenever you regenerate the proxy classes.

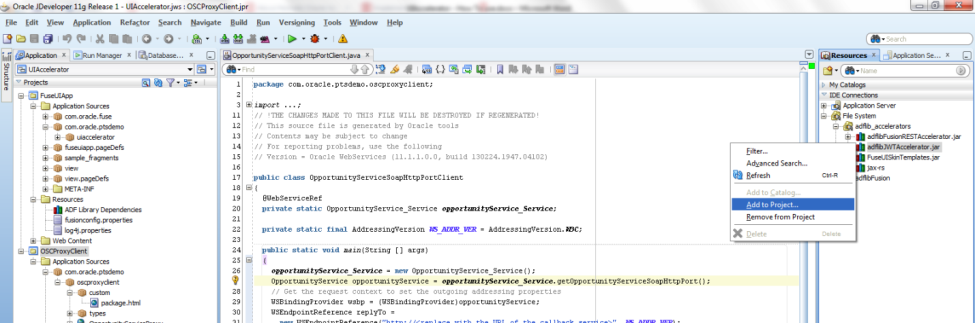


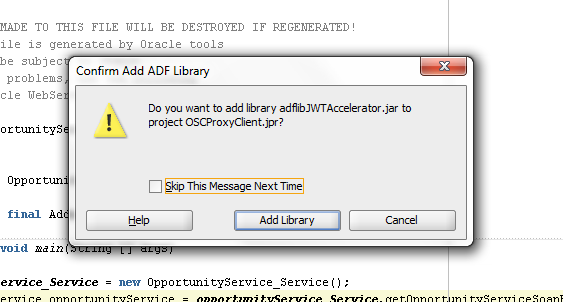
## Using JWT User Token API for Web Service Proxy

In order to use JWT User Token API, as a first step, we will create a new Proxy Class which will contain all the logic to call web service operations. To better identify the new class, add a new package under oscproxyclient package.

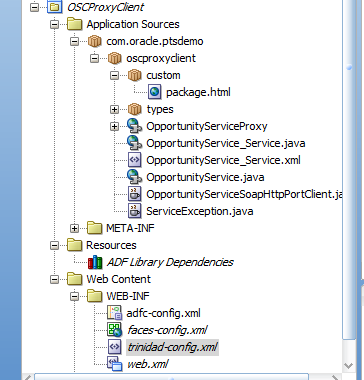


Before creating the class, we need to add the ADF Library adflibJWTAccelerator to the project library. There are several ways to do that. You can create a new File System Connection pointing to the WEB-INF\lib folder of your FuseUIApp project, because libraries in your project are located there.

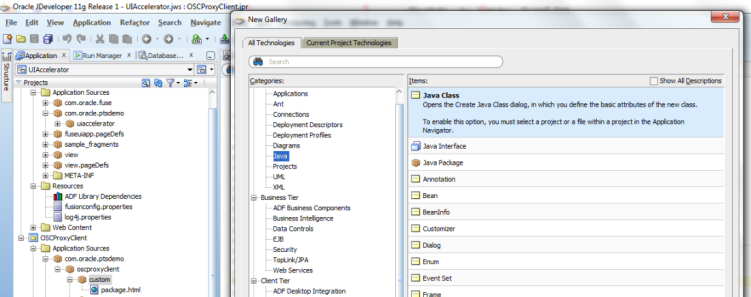
After that, you have to click on OSCProxyClient project, and then, from the Resources Catalog tab, right click on the adflibJWTAccelerator.jar file and select Add to Project: 

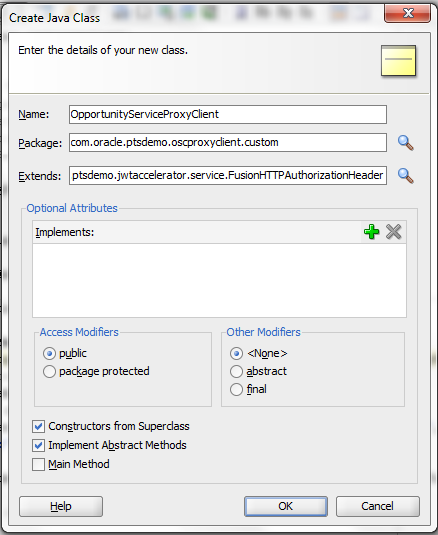


Check that an entry was added into the Resources folder of OSCProxyClient:



Now you can create a new Java Class:





The initial class code should be similar to the content in the code snippet below:

public class OpportunityServiceProxyClient extends FusionHTTPAuthorizationHeader {

public OpportunityServiceProxyClient() {

super();

}

public void initialiseConnection(String p1, String p2, String p3, String p4, int p5, int p6) {

}

}

For some reason, the JDeveloper version used during the development of this sample code was not able to generate the full method’s signature, and because of that, you should not be able to identify the meaning of each method parameter. Despite of that issue, below you can find the full code and all the required steps to build a Java proxy client class, using the JWT User Token API to save you time.

At first, you have to copy the web service reference object (annotated with @WebServiceRef) from the HttpPortClient class automatically generated by JDeveloper to the created class. Then paste it as a private property of the class:

public class OpportunityServiceProxyClient extends FusionHTTPAuthorizationHeader {

@WebServiceRef

private static OpportunityService\_Service opportunityService\_Service;

private OpportunityService opportunityService;

public OpportunityServiceProxyClient() {

super();

}

public void initialiseConnection(String p1, String p2, String p3, String p4, int p5, int p6) {

}

}

Now that you have all the properties assigned to your class, you just need to setup initialiseConnection method. Copy and paste the initialiseConnection method into your class, replacing the original method. The bottom line here is the way that SoapHttpPort method retrieves SecurityPolicy and the need to call the helper function initFusionEndpoint().

/\*\*

\* This method should be used to initialize HTTP Authorization Header

\* @param pJWTToken

\* @param pUsername

\* @param pPassword

\* @param pFusionEndpointURL

\* @param pFetchSize

\* @param pFetchStart

\*/

public void initialiseConnection(String pJWTToken, String pUsername,

String pPassword, String pFusionEndpointURL,

int pFetchSize, int pFetchStart) {

//1.Only Construct it if it needs constructing

if (opportunityService\_Service==null) {

opportunityService\_Service = new OpportunityService\_Service();

}

//1. Retrieve Security Policy (based on JWT User Token or username/password)

opportunityService =

opportunityService\_Service.getOpportunityServiceSoapHttpPort(this.getSecurityFeature(pFusionEndpointURL, pJWTToken));

// Call Helper Function

initFusionEndpoint(pJWTToken, pUsername, pPassword, (WSBindingProvider)opportunityService, pFusionEndpointURL, pFetchSize, pFetchStart);

}

After that, you can create your own methods containing all the business logic. For instance, we will create a method to find all opportunities which are closed and won:

/\*\*

\* Returns a list of WIN-CLOSED Opportunities

\* @param partyUniqueName

\* @throws Exception

\*/

public List<Opportunity> findClosedWonOpportunities(String partyUniqueName) throws Exception {

if (!isInitialised()) {

throw new Exception("Opportunity Service not initialized");

}

//if no value specified, bring all Opportunities

if (partyUniqueName == null || "".equals(partyUniqueName)) {

partyUniqueName = "%";

}

FindOpportunity request = new FindOpportunity();

request.setFindCriteria(buildFindClosedWonOpportunitiesCriteria(partyUniqueName, getFetchStart(), getFetchSize()));

request.setFindControl(buildFindClosedWonOpportunitiesControl());

FindOpportunityResponse response =

opportunityService.findOpportunity(request);

return response.getResult();

}

/\*\*

\* Build FindControl payload section for FindClosedWonOpportunities

\* @return

\*/

private static FindControl buildFindClosedWonOpportunitiesControl() {

FindControl fControl = new FindControl();

fControl.setRetrieveAllTranslations(false);

return fControl;

}

/\*\*

\* Build FindCriteria payload section for FindClosedWonOpportunities.

\* @param partyUniqueName

\* @param fetchStart

\* @param fetchSize

\* @return

\* @see "SOAP\_payload\_find\_CLOSED\_WON\_opty\_by\_SalesParty.xml"

\*/

private static FindCriteria buildFindClosedWonOpportunitiesCriteria(String partyUniqueName, int fetchStart, int fetchSize) {

FindCriteria findCriteria = new FindCriteria();

findCriteria.setFetchStart(fetchStart);

findCriteria.setFetchSize(fetchSize);

//fiter definition

ViewCriteria filter = new ViewCriteria();

filter.setConjunction(Conjunction.AND);

//group definition

ViewCriteriaRow group = new ViewCriteriaRow();

group.setUpperCaseCompare(false);

group.setConjunction(Conjunction.AND);

//item

ViewCriteriaItem partyUniqueNameItem = new ViewCriteriaItem();

partyUniqueNameItem.setConjunction(Conjunction.AND);

partyUniqueNameItem.setUpperCaseCompare(false);

partyUniqueNameItem.setAttribute("PartyUniqueName");

partyUniqueNameItem.setOperator("STARTSWITH");

partyUniqueNameItem.getValue().add(partyUniqueName);

ViewCriteriaItem stageStatusCdItem = new ViewCriteriaItem();

stageStatusCdItem.setConjunction(Conjunction.AND);

stageStatusCdItem.setUpperCaseCompare(false);

stageStatusCdItem.setAttribute("StageStatusCd");

stageStatusCdItem.setOperator("=");

stageStatusCdItem.getValue().add("WON");

//findAttribute

findCriteria.getFindAttribute().add("PartyUniqueName");

findCriteria.getFindAttribute().add("Name");

findCriteria.getFindAttribute().add("OptyId");

findCriteria.getFindAttribute().add("Revenue");

findCriteria.getFindAttribute().add("OptyCreationDate");

findCriteria.getFindAttribute().add("OptyLastUpdateDate");

//sortOrder

SortAttribute partyUniqueNameSortAttr = new SortAttribute();

partyUniqueNameSortAttr.setName("PartyUniqueName");

partyUniqueNameSortAttr.setDescending(false);

SortAttribute optyLastUpdateDateSortAttr = new SortAttribute();

optyLastUpdateDateSortAttr.setName("OptyLastUpdateDate");

optyLastUpdateDateSortAttr.setDescending(true);

SortOrder sortOrder = new SortOrder();

sortOrder.getSortAttribute().add(partyUniqueNameSortAttr);

sortOrder.getSortAttribute().add(optyLastUpdateDateSortAttr);

//assign items to group

group.getItem().add(partyUniqueNameItem);

group.getItem().add(stageStatusCdItem);

//assign group to filter

filter.getGroup().add(group);

//assign filter and sort order to findCriteria

findCriteria.setFilter(filter);

findCriteria.setSortOrder(sortOrder);

findCriteria.setExcludeAttribute(false);

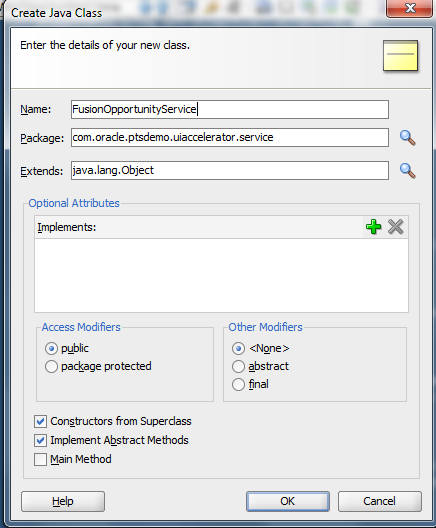
return findCriteria;

}

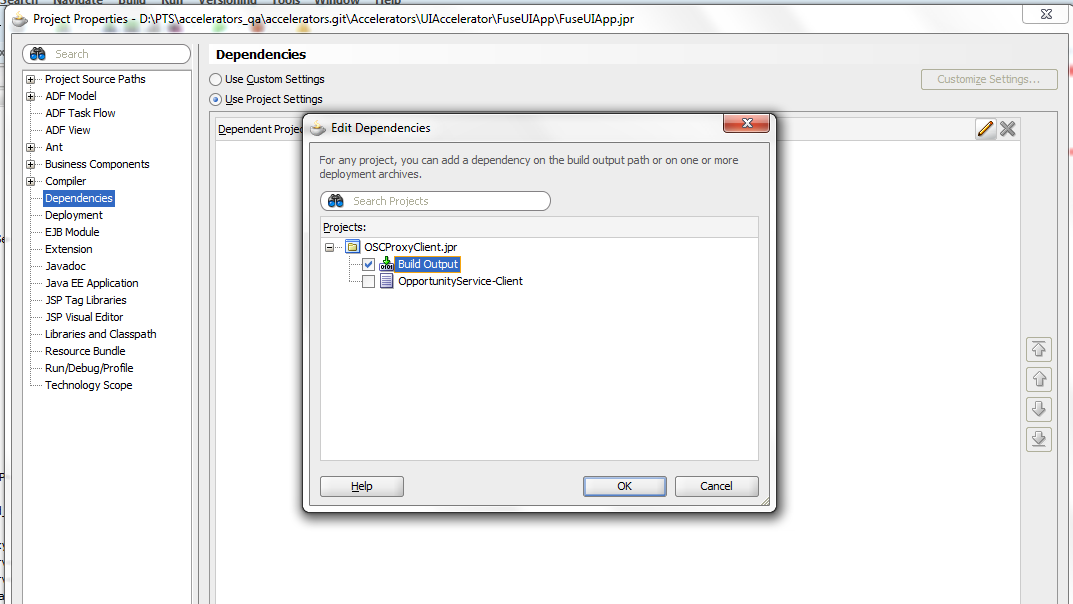
The class resulting on that code is available below. You also can find the xml payload file which has been used to query the required data in OSC.



Finally, you can go back to FuseUIApp project and create the class com.oracle.ptsdemo.uiaccelerator.service.FusionOpportunityService. This is a managed bean object which will be responsible to call the Java proxy client and consume the web service and should act as a Façade. There is also a possibility to expose that business class as an ADF Data Control (Bean Data Control), so that, you can have a chance to drag and drop that one into a page and render it as a table or a another graphical component.



You should also add the OSCProxyClient project into the Project Dependencies of FuseUIApp as displayed below:



Before adding the logic to the OpportunityServiceProxyClient class, create a Bean class OpportunityVO under the package com.oracle.ptsdemo.uiaccelerator.vo. This class should hold the values to display the opportunity into the UI.

Add the following fields and its respective Accessors as displayed below. Don’t forget to right click over fields and select “Generate Accessors…” so that all accessors’ methods should be included into the class:



Next step, you have to go back to FusionOpportunityService.java class and copy the snipped code below into the fresh newly created class:

package com.oracle.ptsdemo.uiaccelerator.service;

import com.oracle.ptsdemo.oscproxyclient.custom.OpportunityServiceProxyClient;

import com.oracle.ptsdemo.oscproxyclient.types.Opportunity;

import com.oracle.ptsdemo.utils.JSFUtils;

import java.util.List;

import java.util.Map;

public class FusionOpportunityService {

private OpportunityServiceProxyClient proxyClient;

public FusionOpportunityService() {

proxyClient = new OpportunityServiceProxyClient();

//read page flow scope variables values

Map<String, String> pHeader = (Map<String, String>)JSFUtils.resolveExpression("#{pageFlowScope.pHeader}");

String pJWTToken = pHeader.get("fusionJWTToken");

String pUsername = pHeader.get("fusionUsername");

String pPassword = pHeader.get("fusionPassword");

String pFusionEndpointURL = pHeader.get("fusionEndpointURL");

int pFetchSize = -1;

if (!"".equals(pHeader.get("fusionFetchSize")) && pHeader.get("fusionFetchSize") != null) {

pFetchSize = Integer.parseInt(pHeader.get("fusionFetchSize")) ;

}

int pFetchStart = 0;

if (!"".equals(pHeader.get("fusionFetchStart")) && pHeader.get("fusionFetchStart") != null) {

pFetchStart = Integer.parseInt(pHeader.get("fusionFetchStart")) ;

}

proxyClient.initialiseConnection(pJWTToken, pUsername, pPassword, pFusionEndpointURL, pFetchSize, pFetchStart);

}

/\*\*

\* Business method responsible to find closed and won opportunities from an specific Party Name

\* @param partyUniqueName

\* @return

\* @throws Exception

\*/

public List<OpportunityVO> findClosedWonOpportunity(String partyUniqueName) throws Exception {

List<Opportunity> list = proxyClient.findClosedWonOpportunities(partyUniqueName);

List<OpportunityVO> response = new ArrayList<OpportunityVO>();

for (Opportunity optyBind : list) {

//The following attributes return in the xml payload

// findCriteria.getFindAttribute().add("PartyUniqueName");

// findCriteria.getFindAttribute().add("Name");

// findCriteria.getFindAttribute().add("OptyId");

// findCriteria.getFindAttribute().add("Revenue");

// findCriteria.getFindAttribute().add("OptyCreationDate");

// findCriteria.getFindAttribute().add("OptyLastUpdateDate");

String partyReturnedUniqueName = optyBind.getPartyUniqueName().getValue();

String optyName = optyBind.getName();

Long optyId = optyBind.getOptyId();

BigDecimal revenueValue = optyBind.getRevenue().getValue();

Calendar optyCreationDate = optyBind.getOptyCreationDate().getValue().toGregorianCalendar();

Calendar optyLastUpdateDate = optyBind.getOptyLastUpdateDate().getValue().toGregorianCalendar();

OpportunityVO optyVO = new OpportunityVO();

optyVO.setPartyUniqueName(partyReturnedUniqueName);

optyVO.setOptyName(optyName);

optyVO.setOptyId(optyId);

optyVO.setRevenueValue(revenueValue);

optyVO.setOptyCreationDate(optyCreationDate);

optyVO.setOptyLastUpdateDate(optyLastUpdateDate);

response.add(optyVO);

}

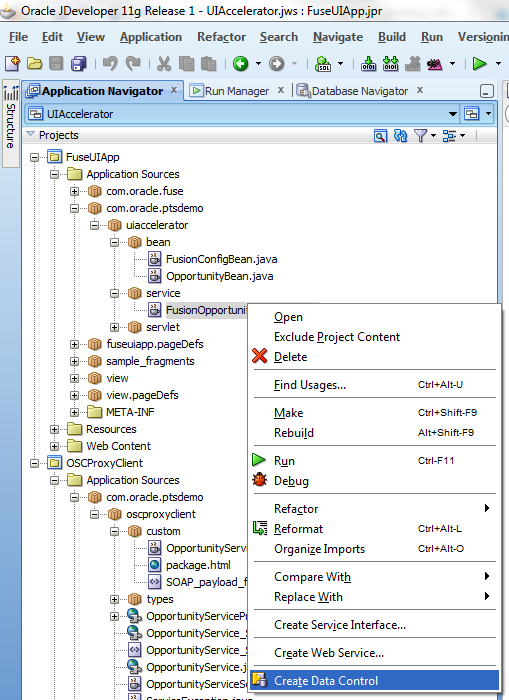
return response;

}

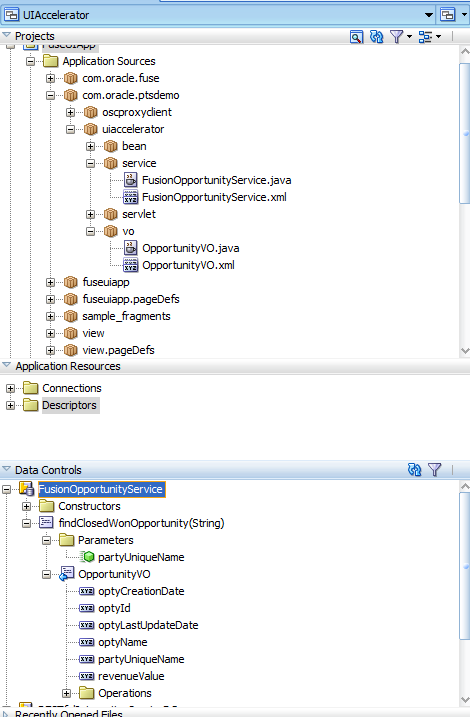
}

## Creating a Bean Data Control to expose Web Service operations

After creating the class, we can expose that as an ADF Data Control. Right Click over the FusionOpportunityService class and select Create Data Control:

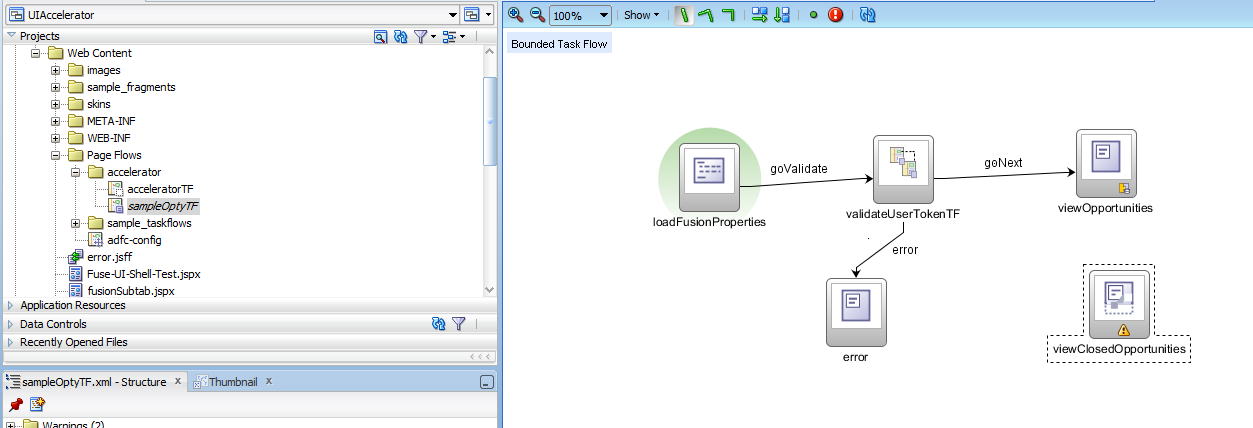


Note that two xml files were created below related business classes with exactly the same name of those classes, and also a new entry was added into the Data Controls section:

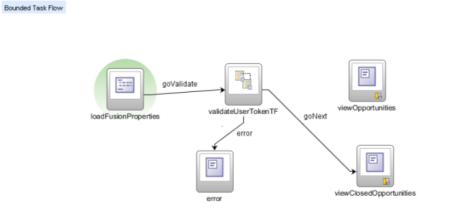


## Building the UI

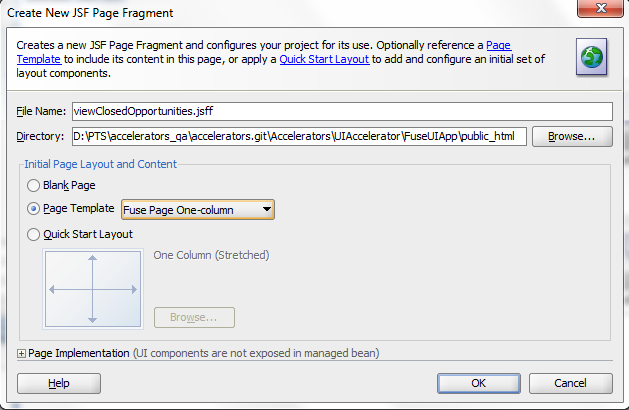
Open up the sampleOptyTF bounded task flow located under Web Content\Page Flows\accelerator folder and drag and drop a new View activity into the Task Flow. Rename the file to “viewClosedOpportunities”:



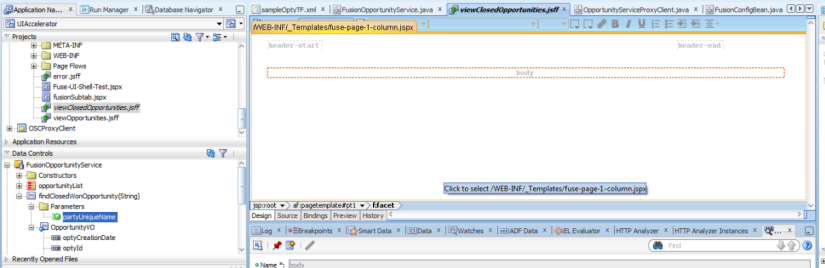
Then, move the goNext Control Flow Case arrow to viewClosedOpportunities view.



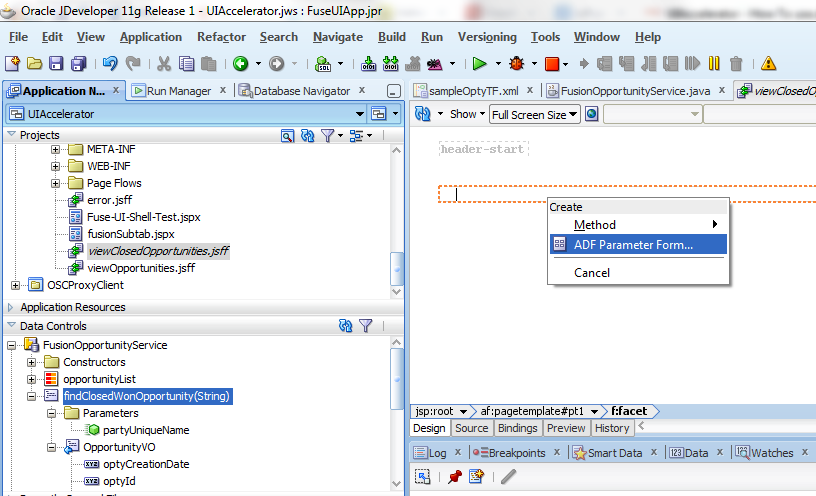
Double click on viewClosedOpportunities icon in order to create a new JSF Page Fragment. On the next window, leave the File Name and Directory fields with the default value. And then, select on Page Template, the option “Fuse Page One-Column”. Press OK to confirm your options.



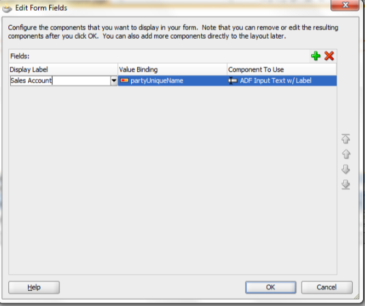
A blank page with no content is displayed:



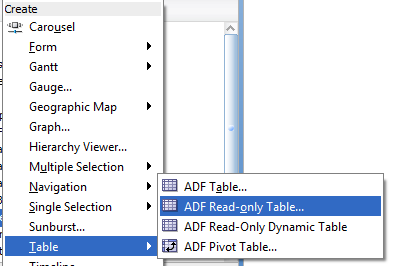
Drag and Drop findClosedWonOpportunity(String) method from the FusionOpportunityService data control into the body section of your page. Click on Create “ADF Parameter Form…” on the popup window:



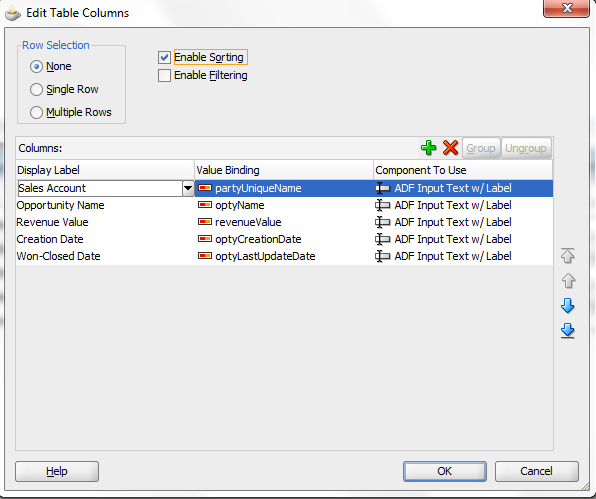
On Edit Form Fields Window, update the Display Label value to “Sales Account” and click OK:



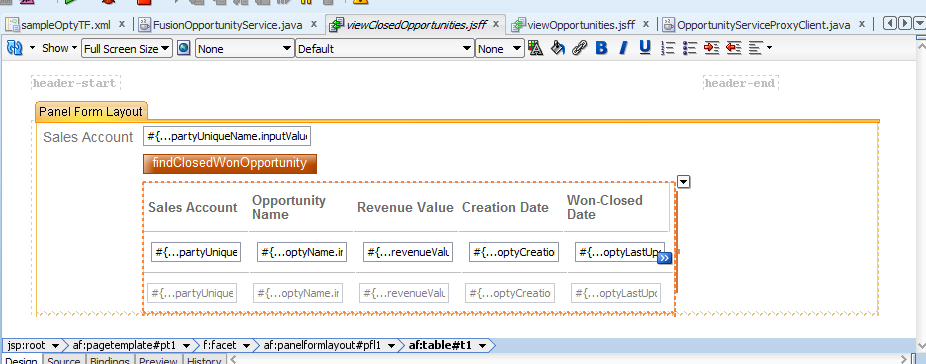
Now, drag and drop OpportunityVO method return object within findClosedWonOpportunity(String) to right after the findCloseWonOpportunity button, as an ADF Read-only Table:



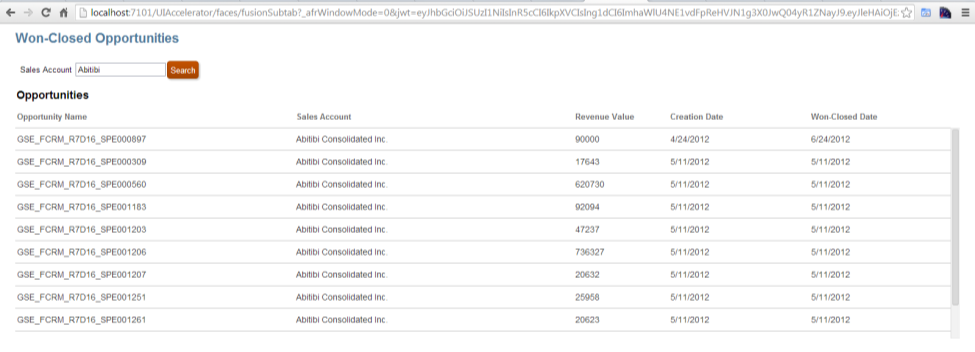
Rename and Organize columns as per the image below:

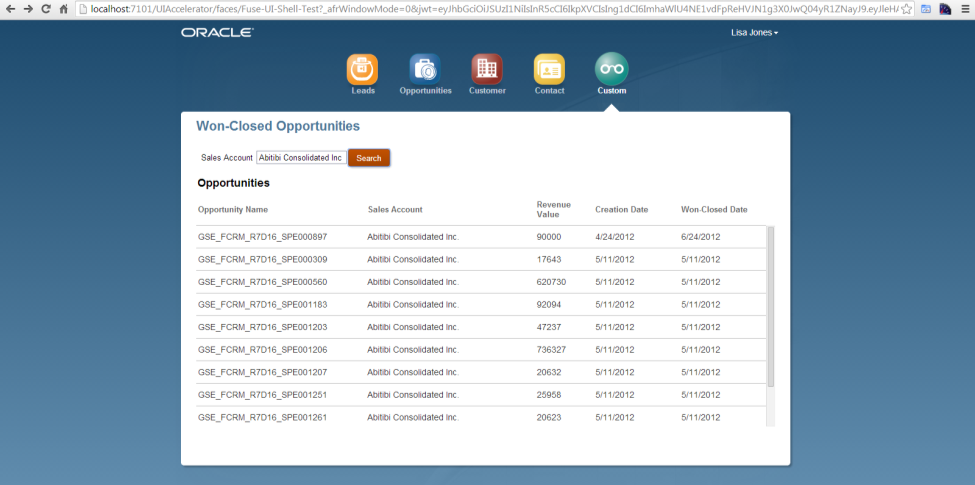


The created page should looks like the one below:

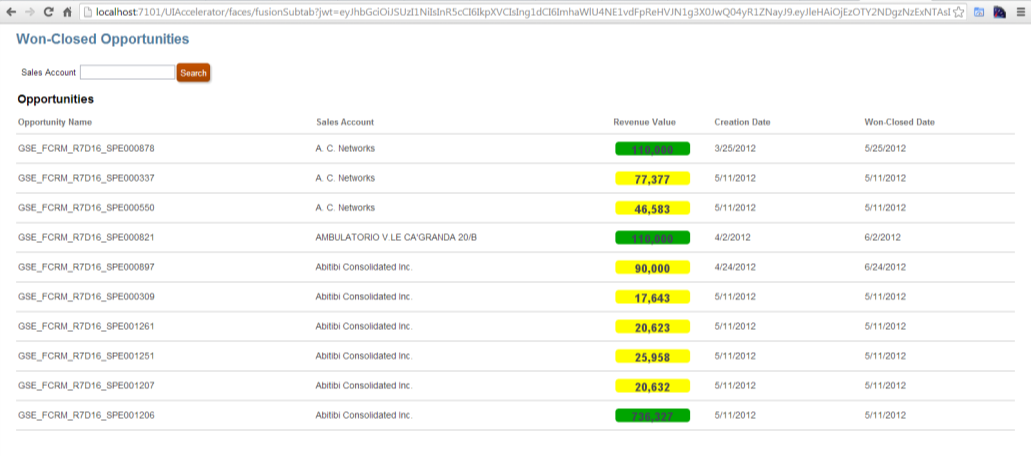


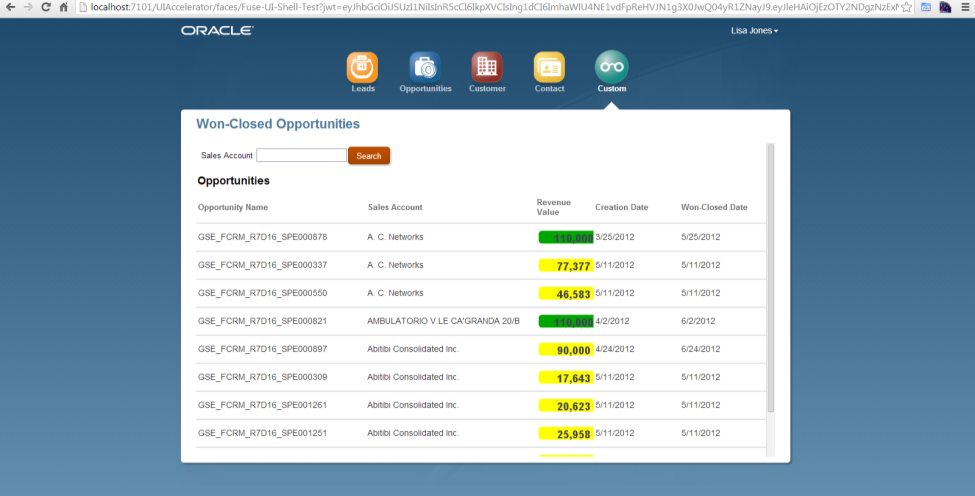
Change the Layout of the page based on the viewOpportunities.jsff and all other pages available on this project to make that page fancy.





You can also add dvt graphs into your page, for instance, replacing the Revenue Value by an ADF DVT Gauge:





The snipped code below can be used to replace the content within af:column (output text) component by the the dvt gauge:

<af:column sortProperty="#{bindings.OpportunityVO.hints.revenueValue.name}"

sortable="true" headerText="Revenue Value" id="c4"

width="10%" minimumWidth="30">

<dvt:gauge id="gauge1" gaugeType="LED" shortDesc="Revenue Value"

imageWidth="200" imageHeight="50" minValue="0" maxValue="1000000"

value="#{row.revenueValue}" ledStyle="LS\_RECTANGLE"

visualEffects="NONE" animationIndicators="NONE"

thresholdDialStyle="TDS\_PIE\_FILL"

gaugeSetAlignment="GSA\_CENTER"

inlineStyle="width:120px; height:25px;">

<dvt:gaugeBackground>

<dvt:specialEffects/>

</dvt:gaugeBackground>

<dvt:thresholdSet>

<dvt:threshold fillColor="#ff0000" thresholdMaxValue="10000"/>

<dvt:threshold fillColor="#ffff00" thresholdMaxValue="100000"/>

<dvt:threshold fillColor="#00a500" />

</dvt:thresholdSet>

<dvt:topLabel/>

<dvt:bottomLabel/>

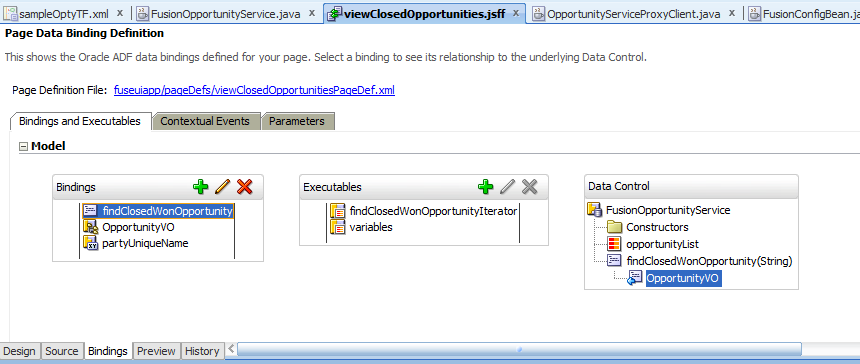
<dvt:metricLabel position="LP\_INSIDE\_GAUGE"

scaling="none"/>

</dvt:gauge>

</af:column>

Finally, the binding definition page for the viewClosedOpportunities.jsff should contains the following objects:



## Testing the Application

Review the properties on files fusionconfig.properties and FusionConfigBean.java, check project project’s dependency setting and follow the same steps used to deploy the application previously. Since we have just replaced viewOpportunity page fragment by viewClosedOpportunities one, there is no need to change any file descriptor or groovy script. Deploy it to WebLogic/Cloud and open up your application.

Note: In case you started facing a security error related to file access to path “.” when calling OSC web service operation from JCS. Comment the static block of the code which has a reference to new File(“.”).toURL(). Below there is a snippet code of the class “\_Service.java” ( Opportunity\_Service.java) which needs to be commented:

static

{

try

{

logger = Logger.getLogger("com.oracle.ptsdemo.oscproxyclient.OpportunityService\_Service");

URL baseUrl = OpportunityService\_Service.class.getResource(".");

if (baseUrl == null)

{

wsdlLocationURL =

OpportunityService\_Service.class.getResource("https://<my oracle sales cloud server>/opptyMgmtOpportunities/OpportunityService?wsdl");

if (wsdlLocationURL == null)

{

baseUrl = new File(".").toURL();

wsdlLocationURL =

new URL(baseUrl, "https://<my oracle sales cloud server>/opptyMgmtOpportunities/OpportunityService?wsdl");

}

}

else

{

if (!baseUrl.getPath().endsWith("/")) {

baseUrl = new URL(baseUrl, baseUrl.getPath() + "/");

}

wsdlLocationURL =

new URL(baseUrl, "https://<my oracle sales cloud server>/opptyMgmtOpportunities/OpportunityService?wsdl");

}

}

catch (MalformedURLException e)

{

logger.log(Level.ALL,

"Failed to create wsdlLocationURL using https://<my oracle sales cloud server>/opptyMgmtOpportunities/OpportunityService?wsdl",

e);

}

}