# AMIS SIG – Low Code development with VBCS

September 2018

In this workshop you will experience Low Code development with Oracle Autonomous Visual Builder (Cloud Service). You will create a Business Object by simply dragging and dropping an Excel file. You will create an application with the created Business Object. You will explore the things VBCS creates for you. You will create an editable table, which requires just a bit of code.

In this SIG you get a rough idea what VBCS can do and what it feels like. Hopefully it will make some of you curious!

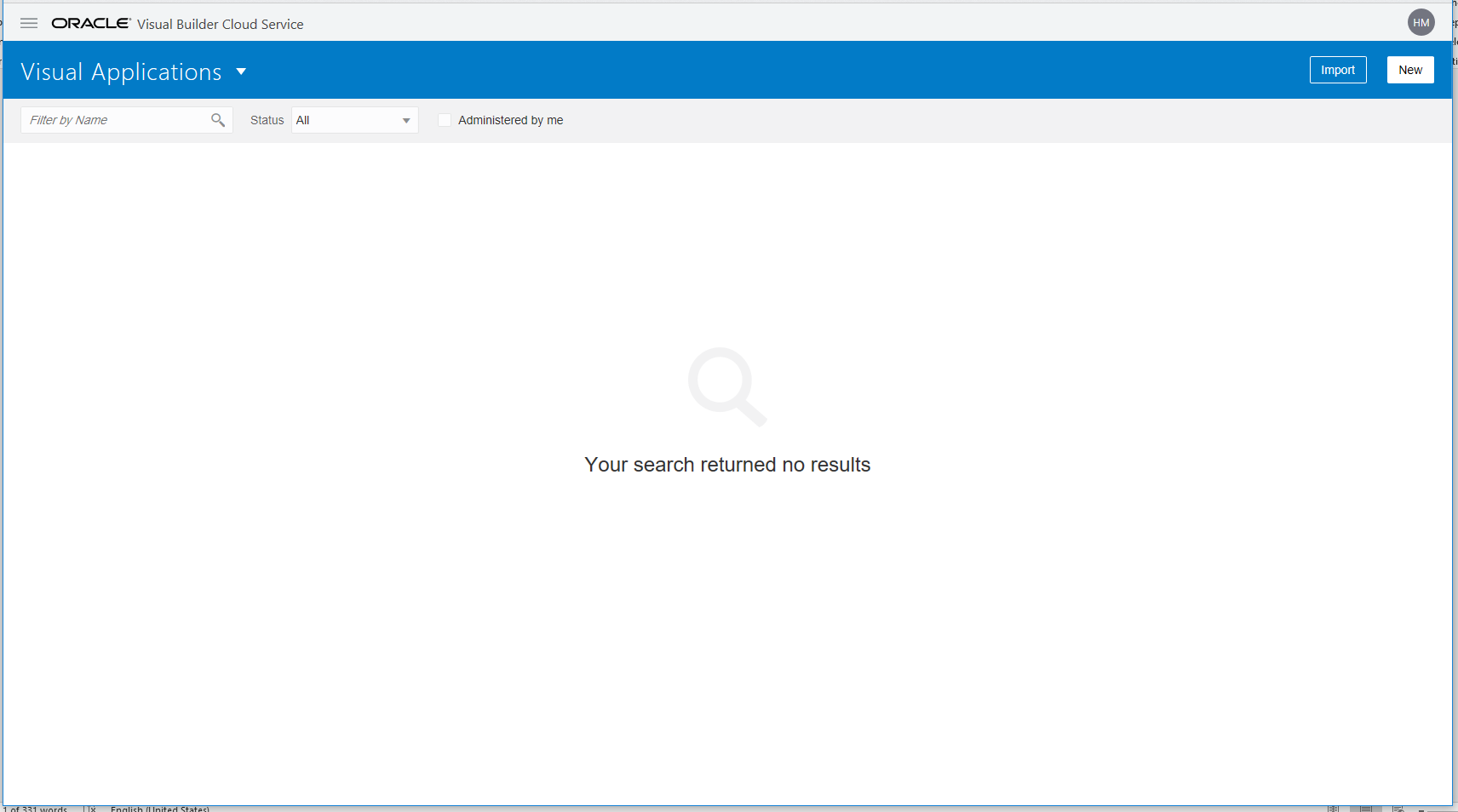
Starting VBCS

In this SIG you can use the VBCS service already created. But there is no problem in creating your own trial via <https://myservices.us.oraclecloud.com/mycloud/signup>

A step by step instruction can be found at the end of this document.

<https://visualbuilderhmensingqsvb-hmensing.builder.ocp.oraclecloud.com/ic/builder/>

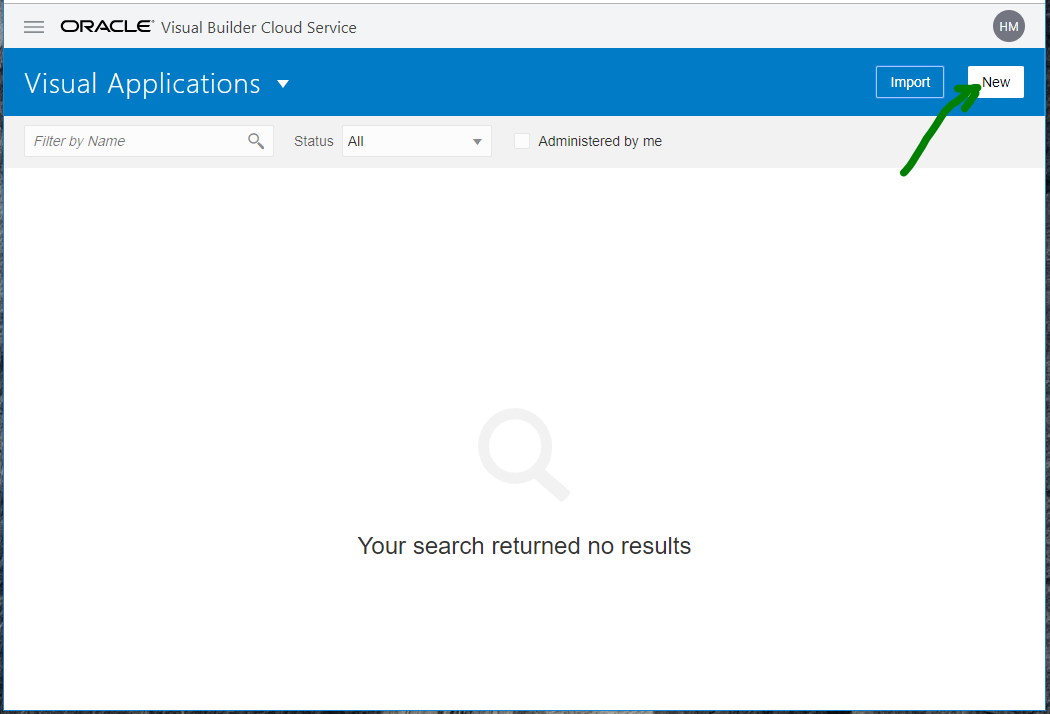
You users will be created by me. You will receive an email from which you can set a password. After that you should get this page.



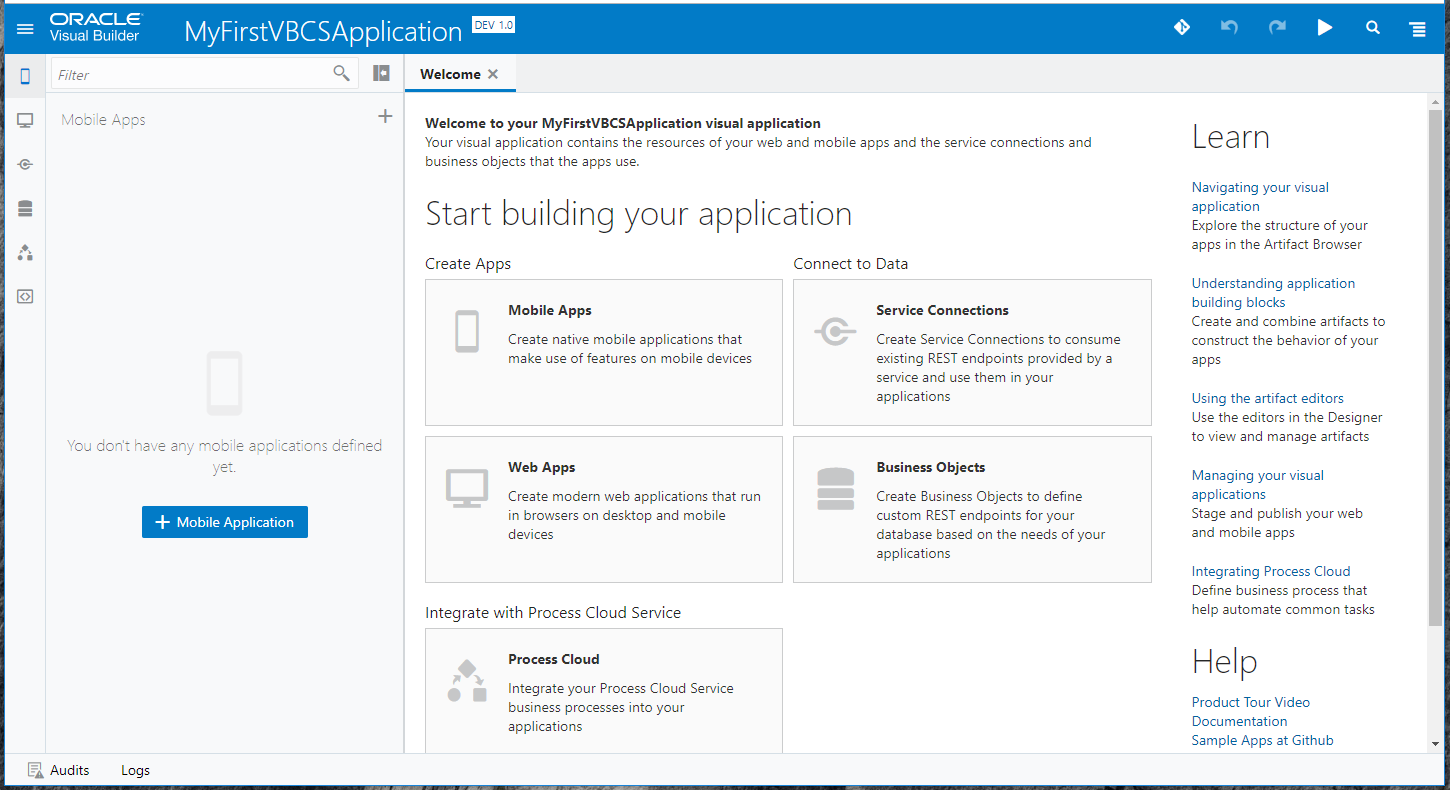
Any file needed for this SIG can be found in Github: https://github.com/AMIS-Services/VBCS-SIG

### Create Application

Create an application, name it whatever you like.







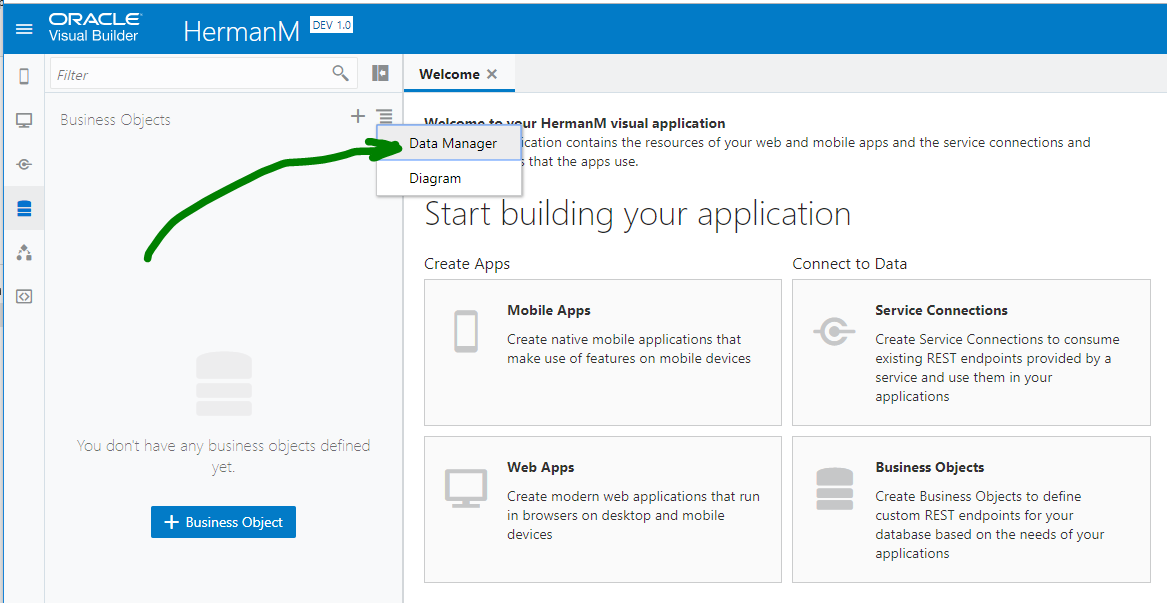
An application can be created from 5 types of resources. In general you will at least have an App (Mobile and/or Web).

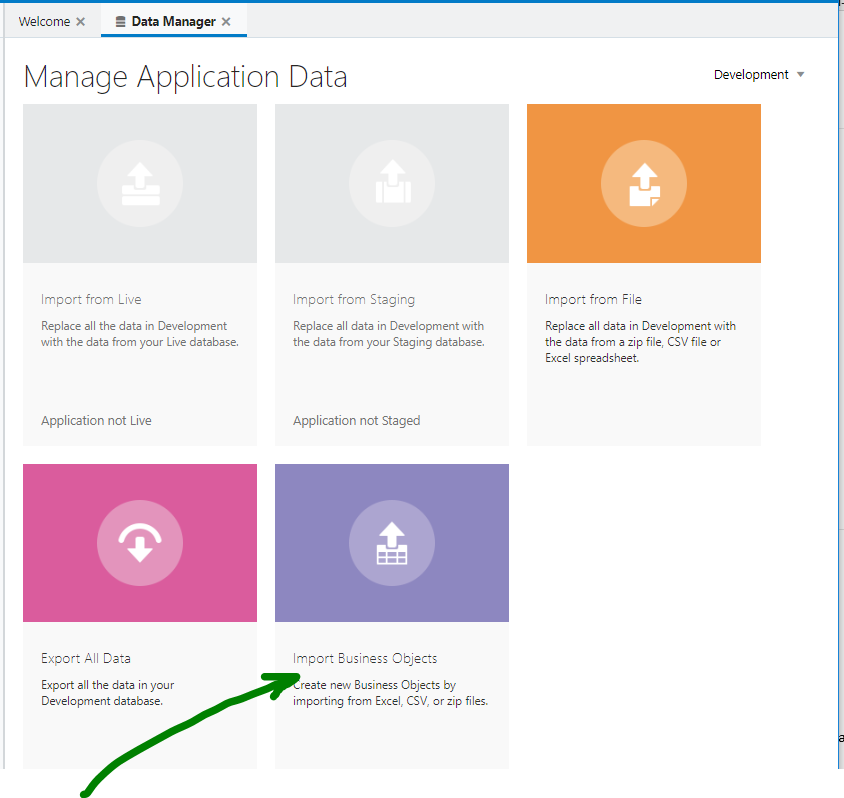
1. Mobile App – Android or iOS
2. Web App – for modern browsers
3. Service Connection – REST API
4. Business object – Table in Oracle database with REST API
5. Process Cloud Service – direct integration with PCS

In this SIG we will start with creating Business Objects and build a Web App on it.

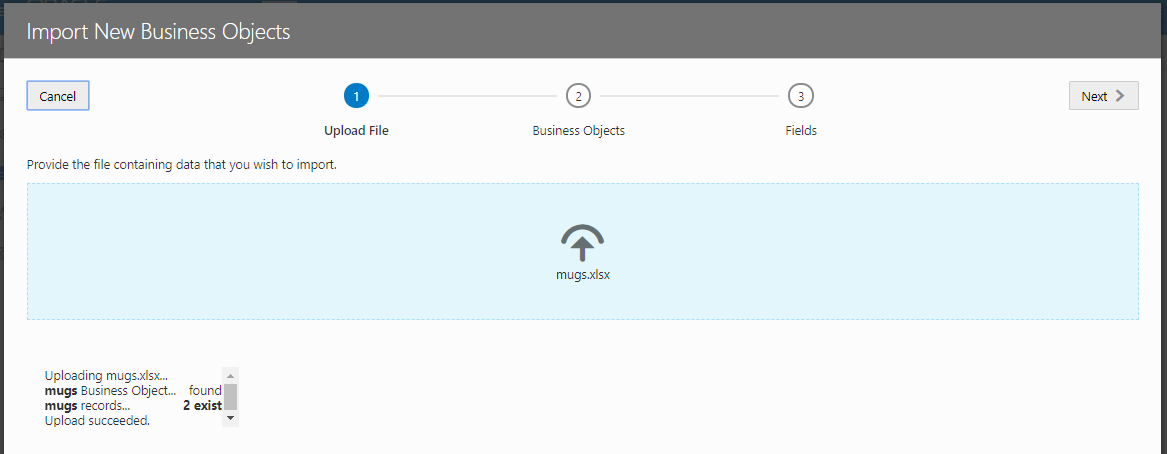
### Create Business Object

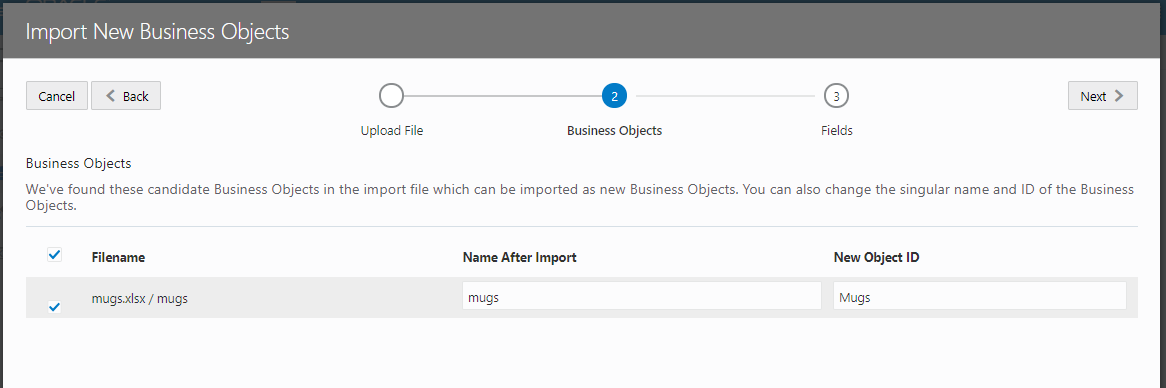
We will create one Business Objects from one Excel-file.

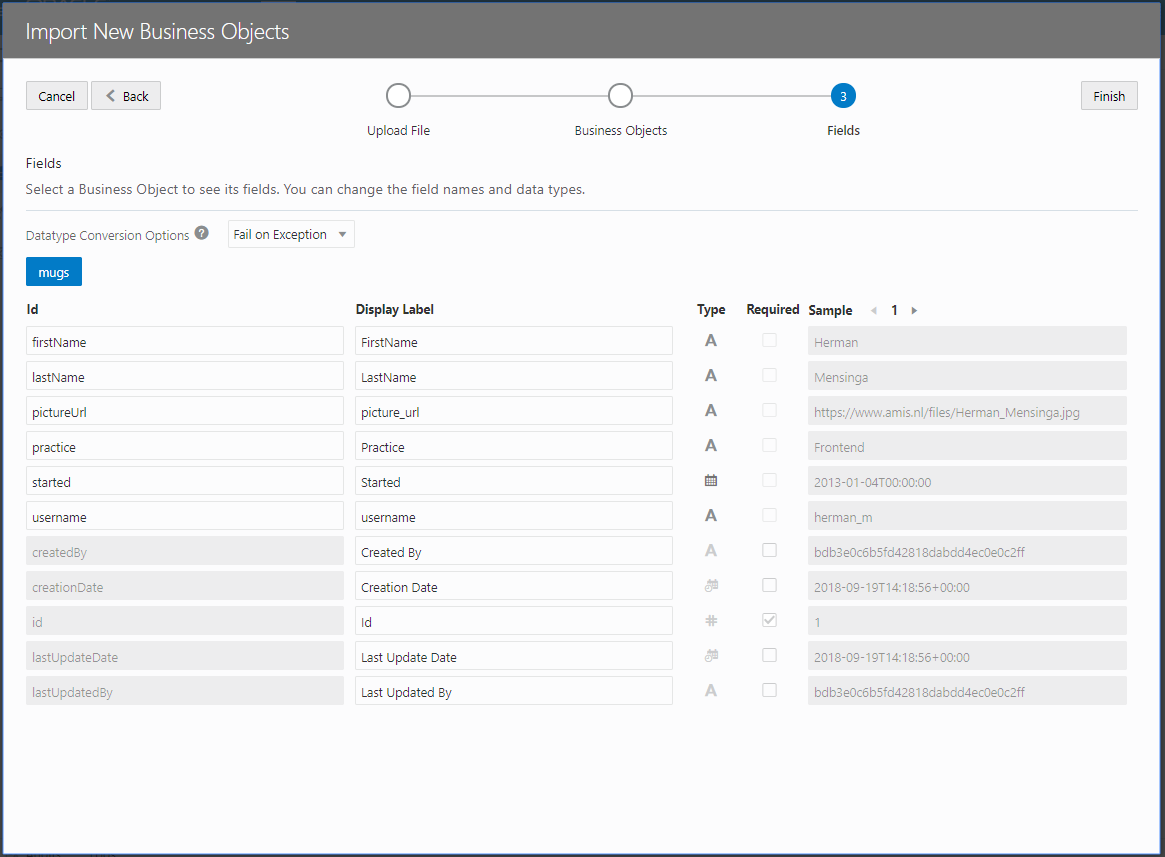




Drag and drop the file mugs.xlsx (in Dutch smoel.xlsx)

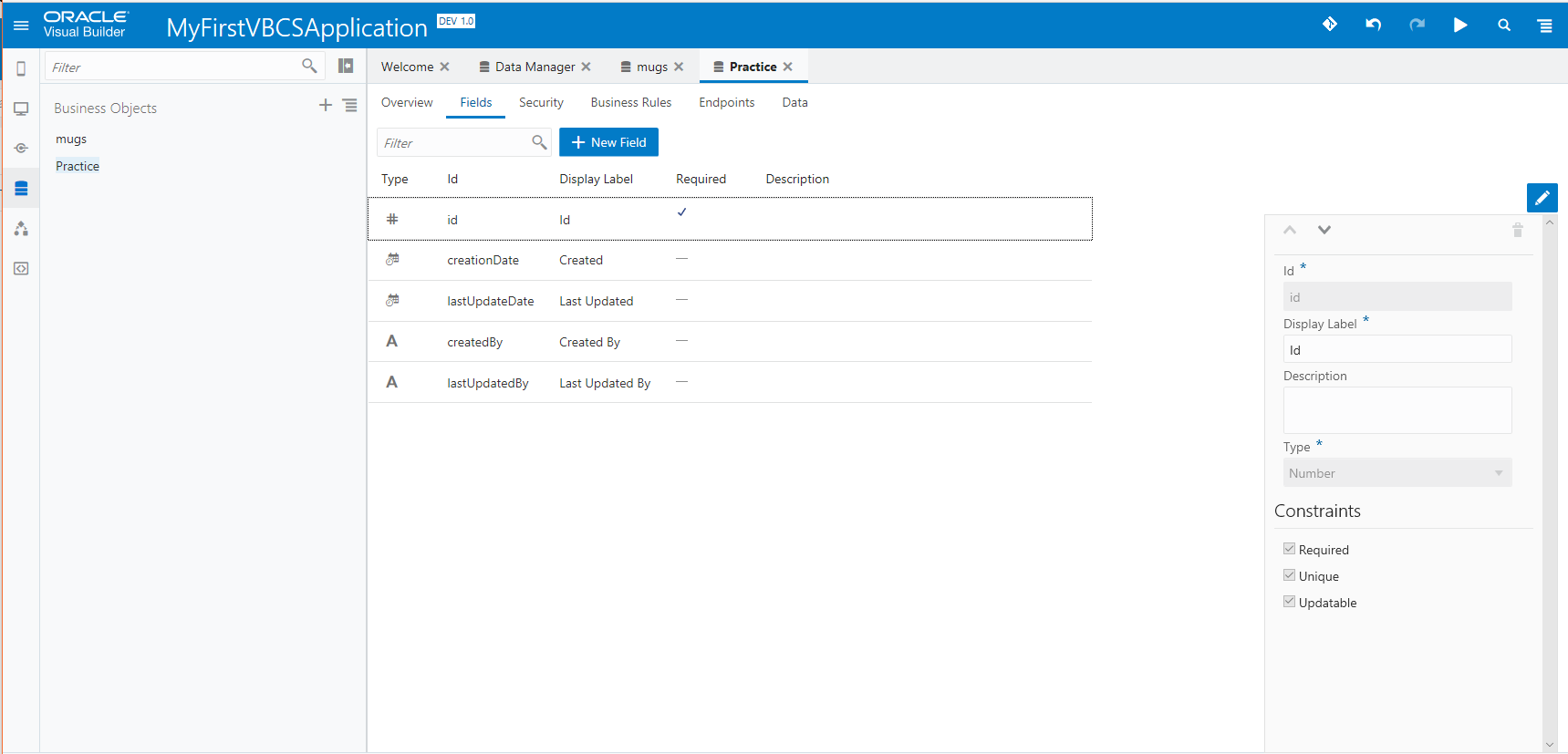




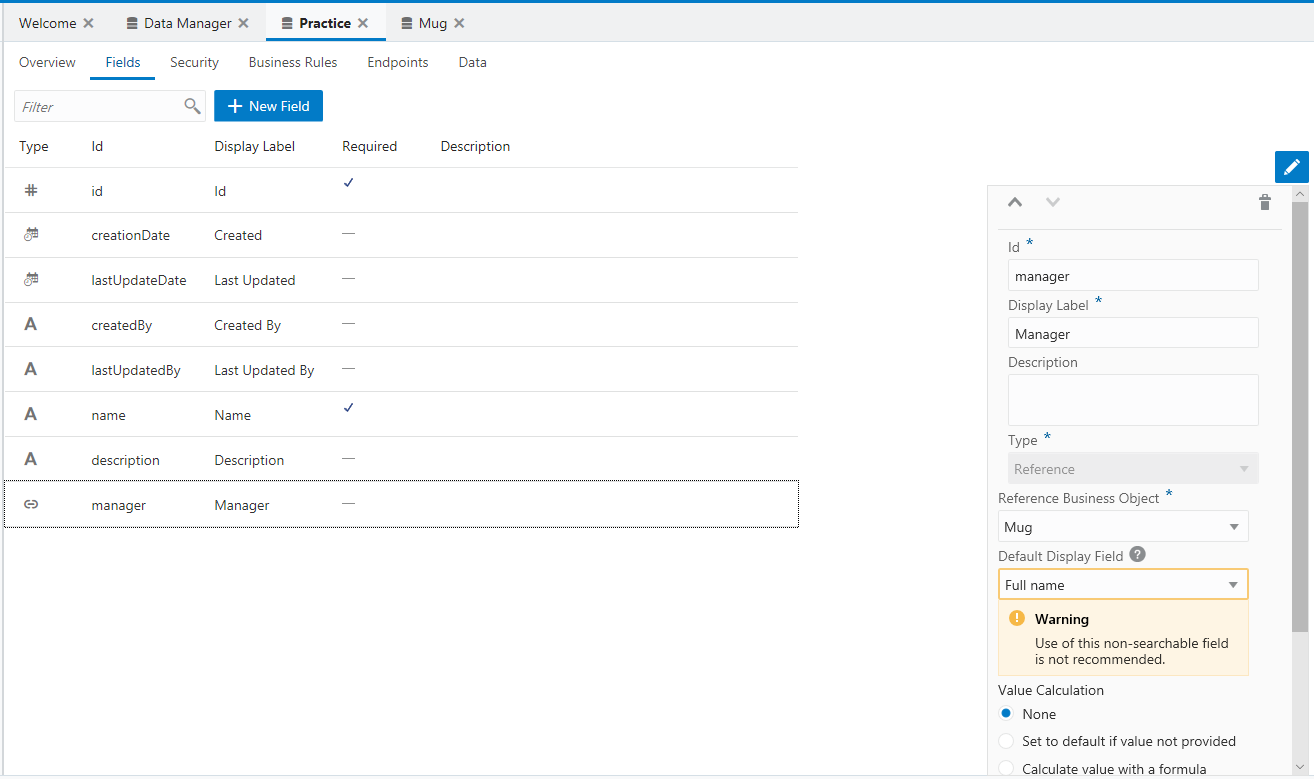


Validate the created BO. Validate the column data types, give them a nice Display label.

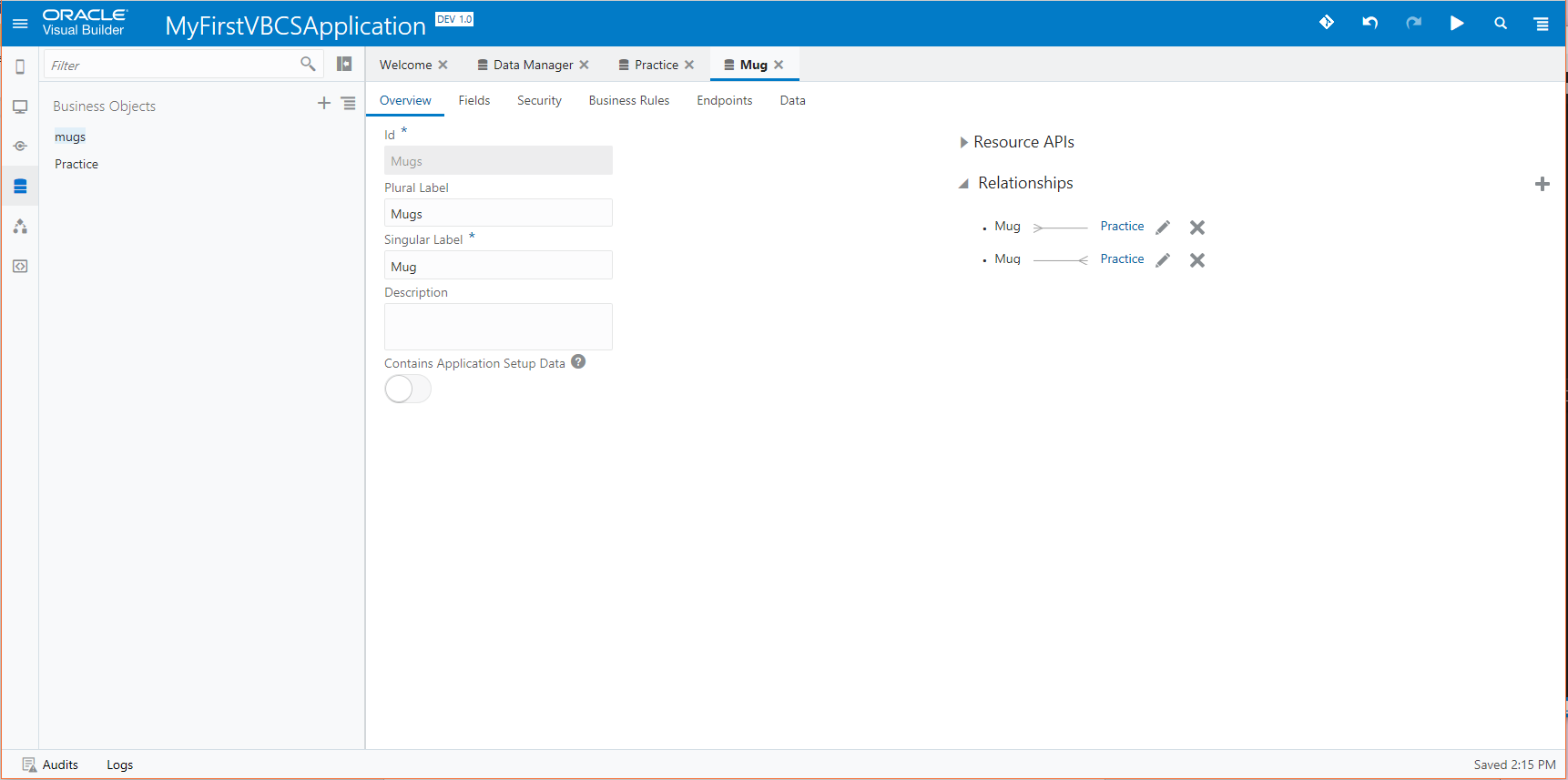
The second BO we will create manually. It will be the Practices BO, with a name, description and manager (which will be a link to the Mugs BO we just created).



Notice that VBCS creates 5 columns by default with triggers to populate them.



Finally we will create a Link to the practices BO from Mugs BO.



### Create a Web Application

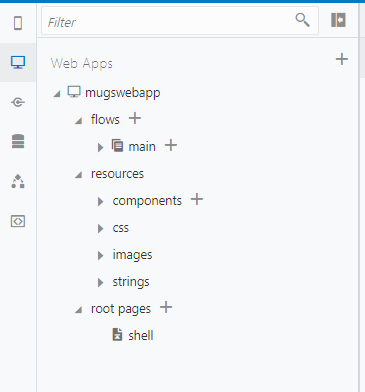
We could also create a Mobile Application, but for this SIG creating a Web Application is quicker and gives us enough opportunities to explore some of the concepts.



#### Created structure

When you create a WebApp, you get a directory containing:

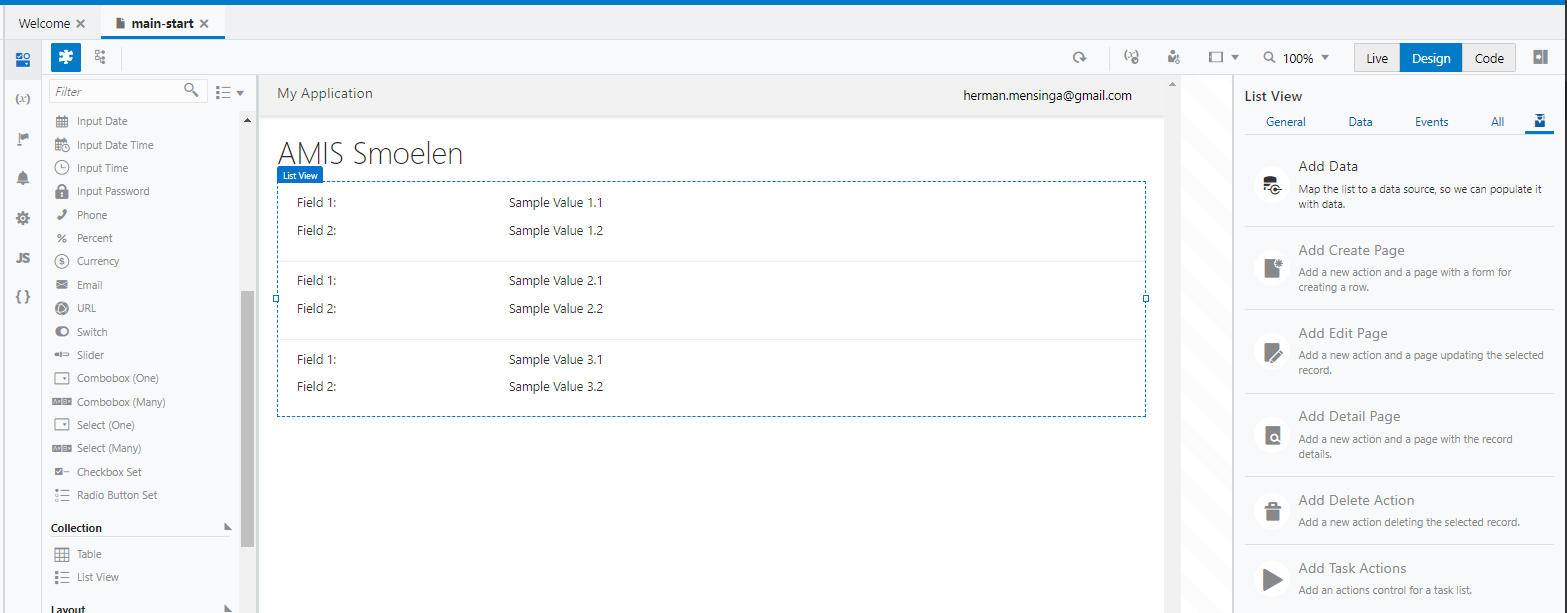
* Flows, every page is part of a flow. The flow main is created by default;
* Resources, for resources like components, css and images
* Root pages, all pages not being part of a flow like the shell page



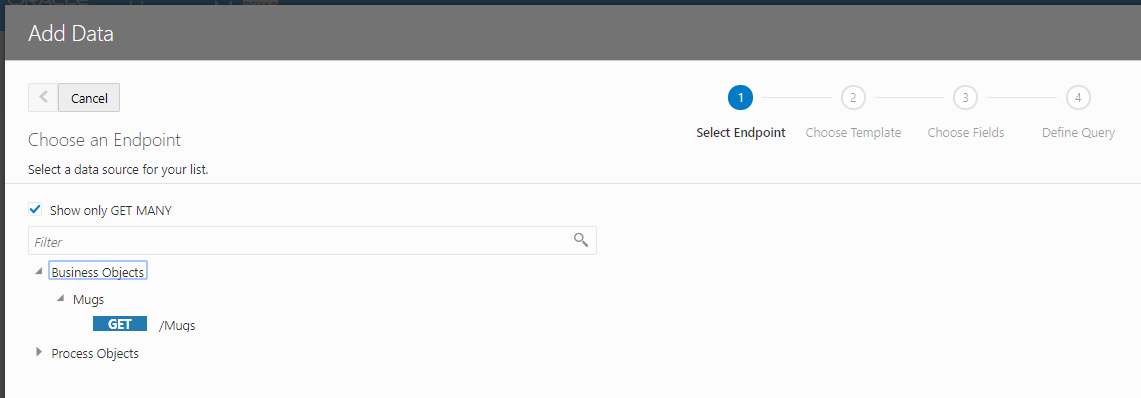
Open the main-start page within the main-flow and drag a Heading on it.

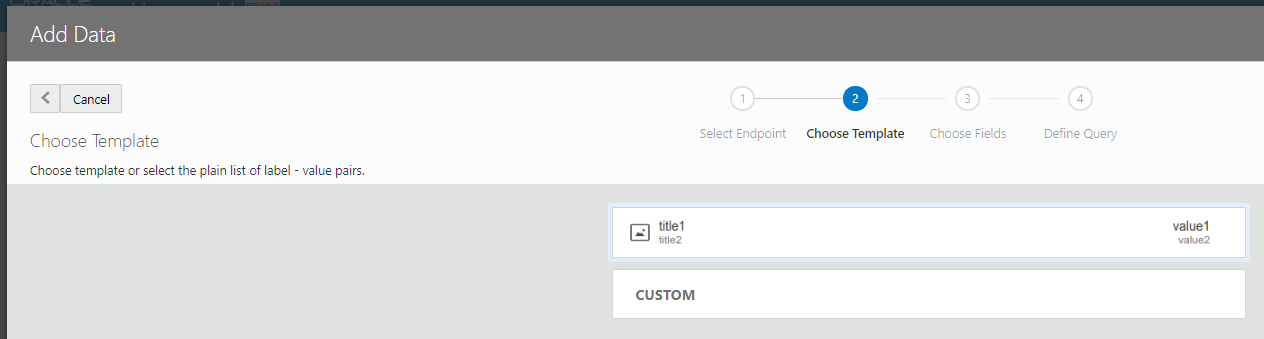
#### Create the overview page

Next we add a List View to the main-start page.

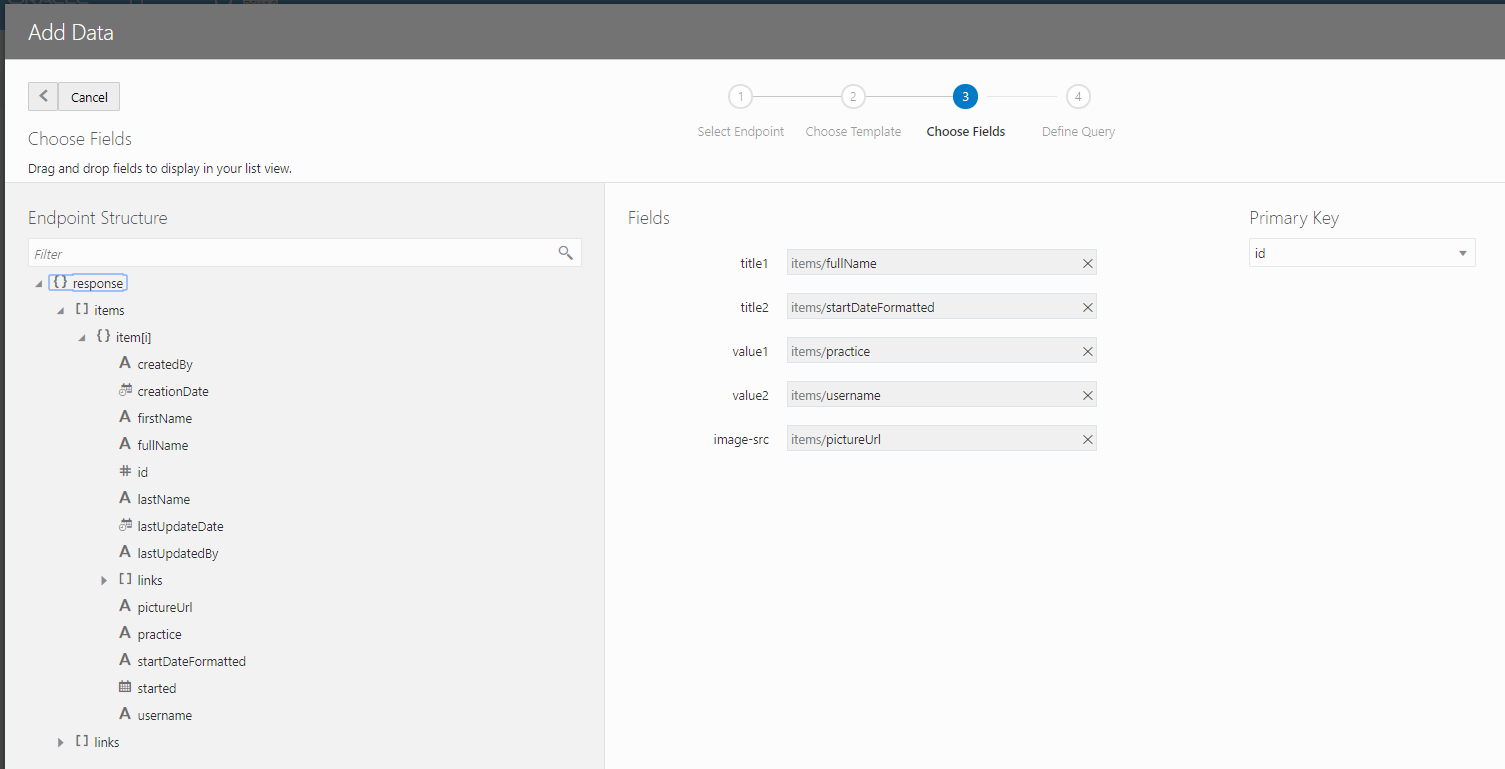


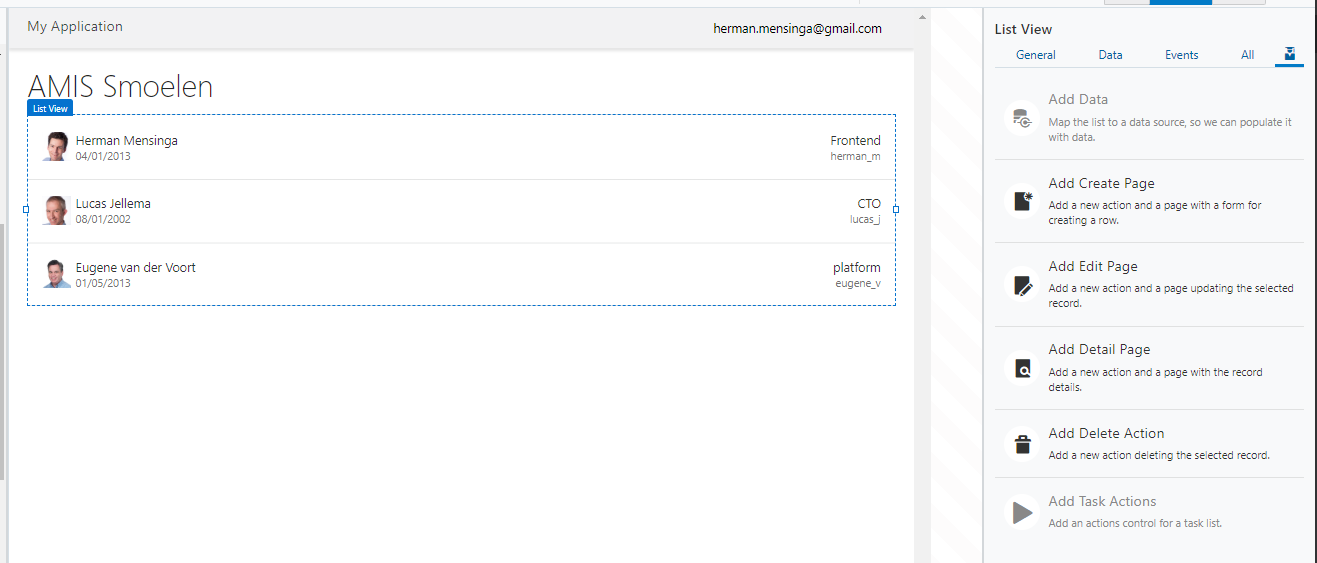
Add the created Mugs Business Object





Drag and drop the information you want to show





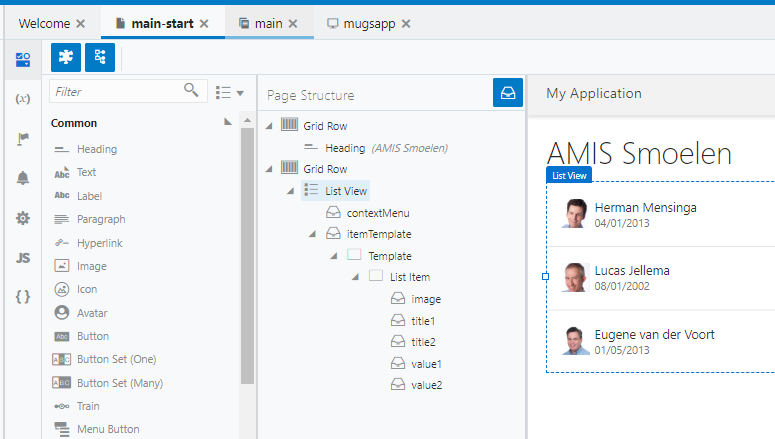
If you open the Code Tab you see that we have created HTML5 with Oracle Jet components/tags.

There is also a ()variable created for the Data, that variable uses an Endpoint and a Type.

Creation of a Table would be comparable to the creation of a List View.

#### Add a Create/Edit and detail page

Like we added Data using a Quick Start, we can also add a Create, Edit and/or Detail Page. If you do not have the List View selected, you can select it easily from the Page Structure ().



Each page will result in a Page being created, a button with Action and some variables. Explore what is created for the four pages.

Could you answer these questions:

* What is this ETag doing in the Edit Page?
* What is the SelectedId variable doing in the main-start Page?
* Where is the MugsId passed in the Edit and Detail Page?

Use a REST API/interface

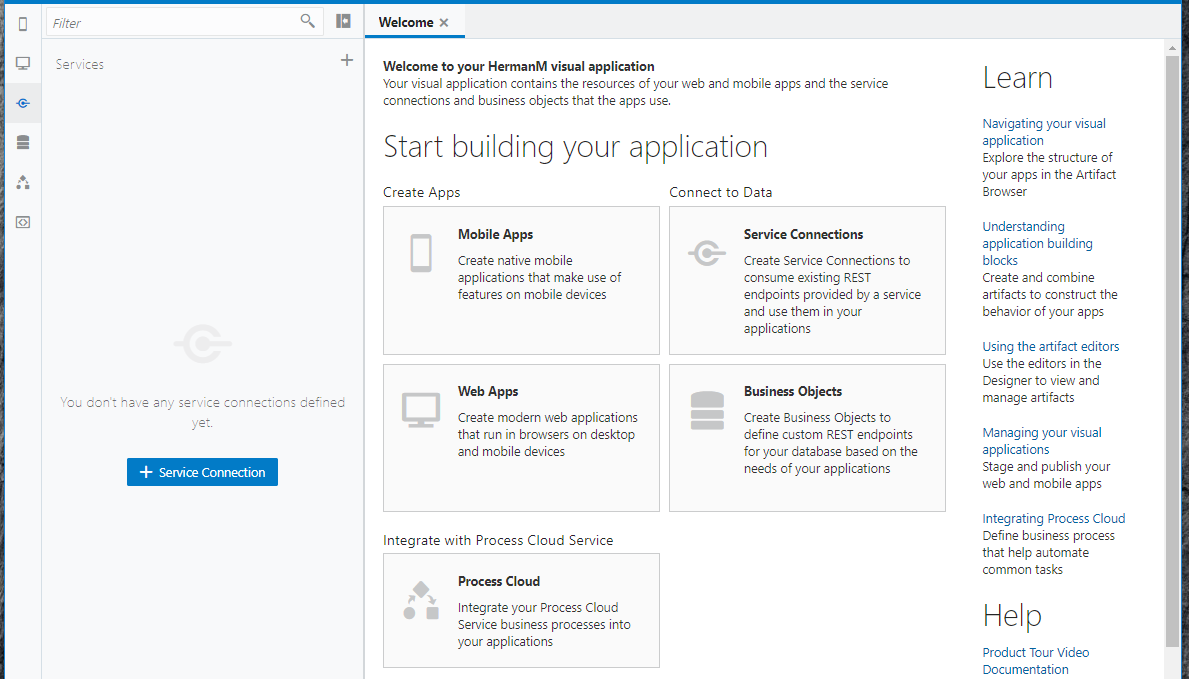
We will make use of the REST Countries REST API, <https://restcountries.eu/rest/v2/>

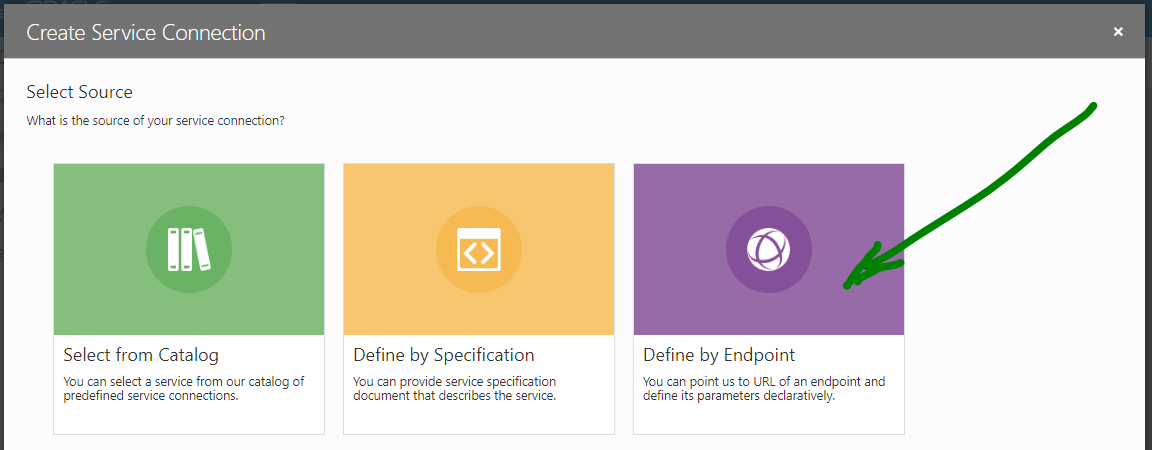
We will add a selection for favorite country to the Mugs page.

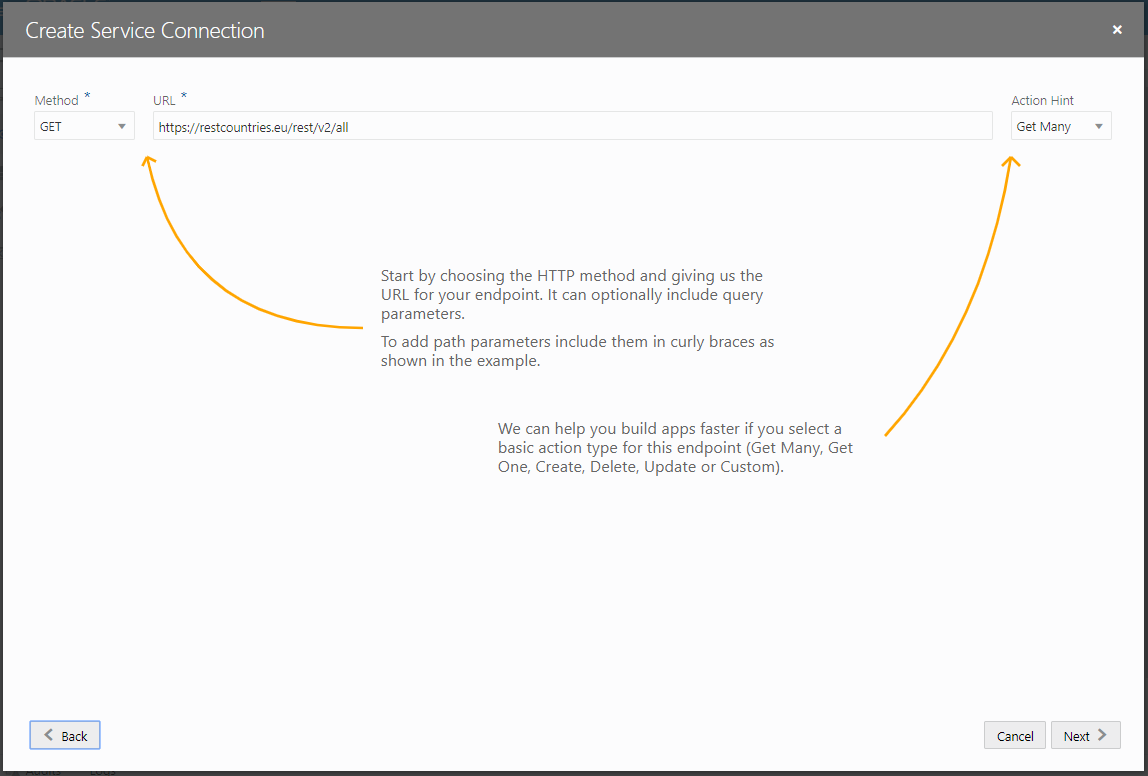
In this blog you find a video explaining and showing most of what we will do here.

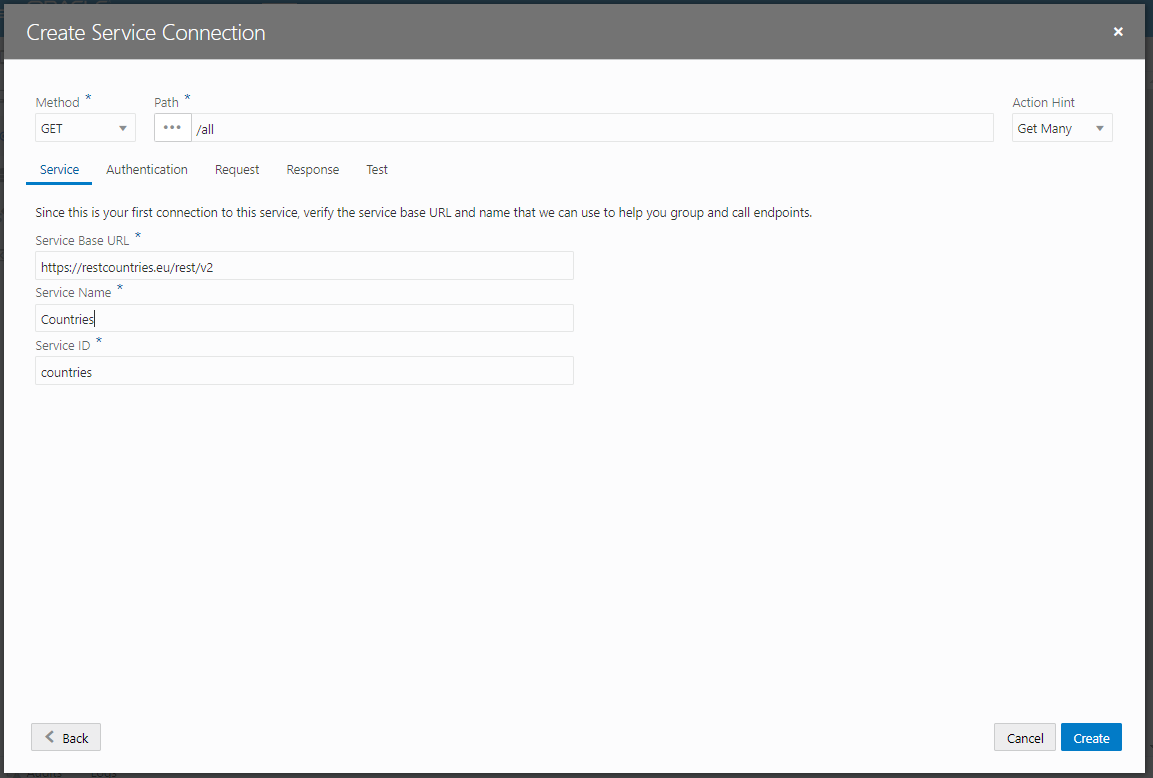
#### Create an Endpoint

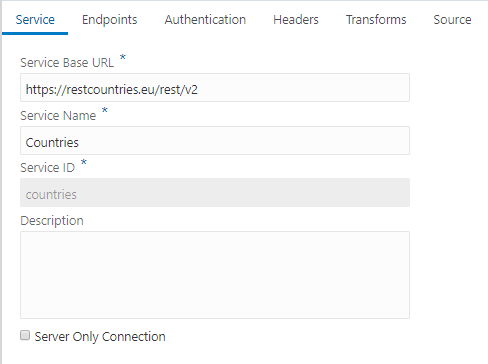
The list of available countries we will pick from the mentioned REST API.

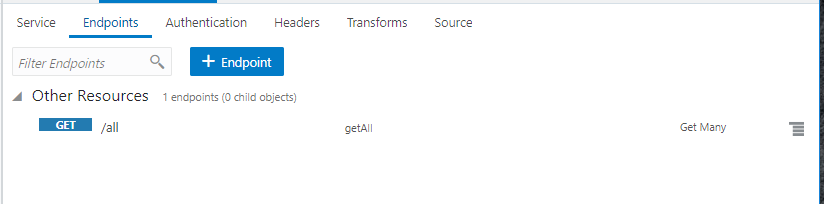








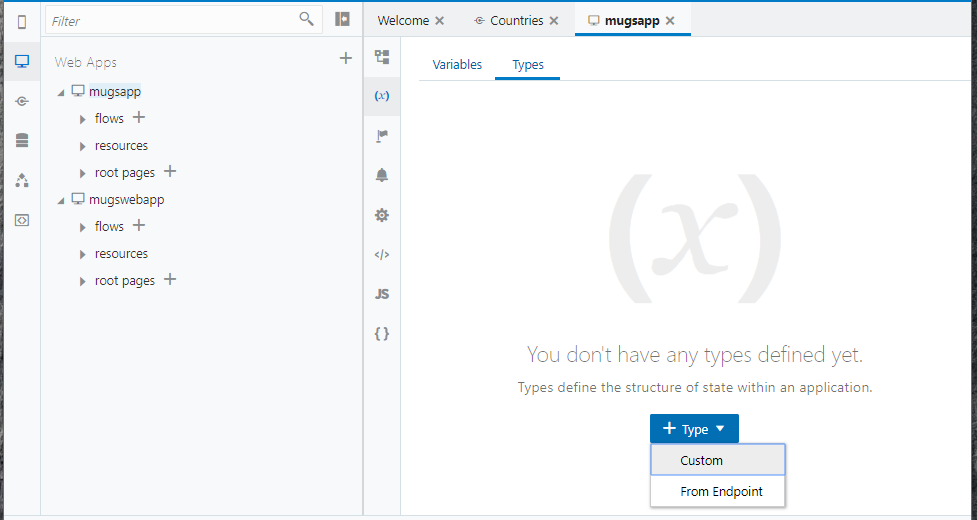




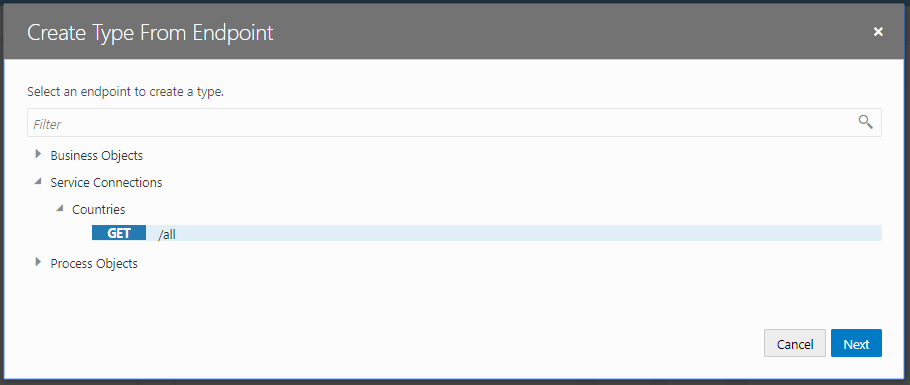
Notice that a Service is created with one Endpoint. A Service can contain multiple (types of) Endpoints.

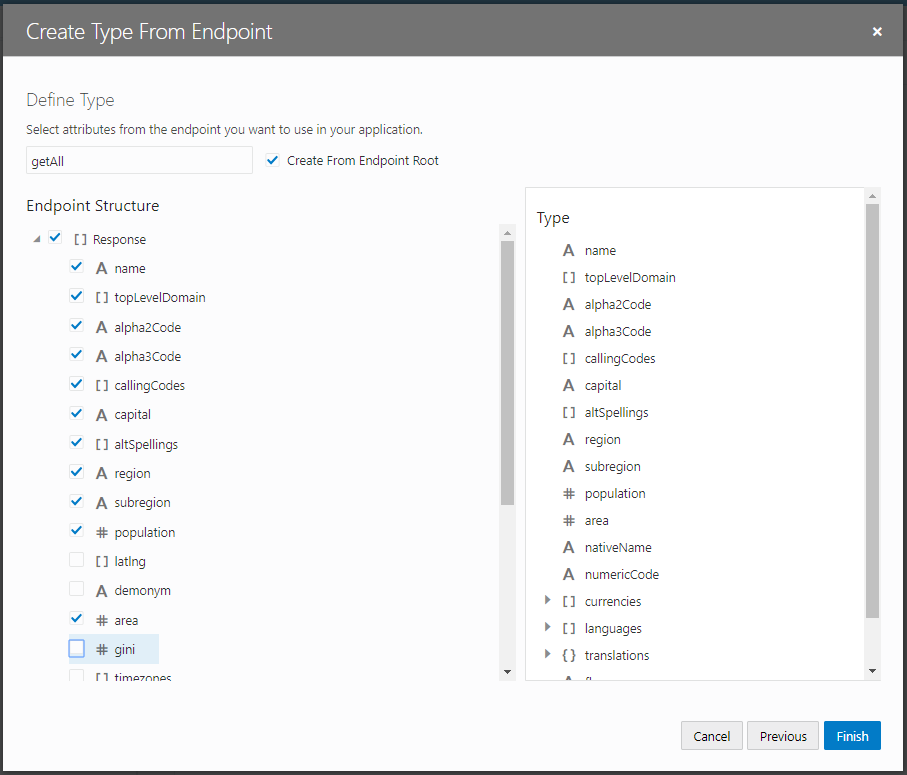
#### Create types and variables

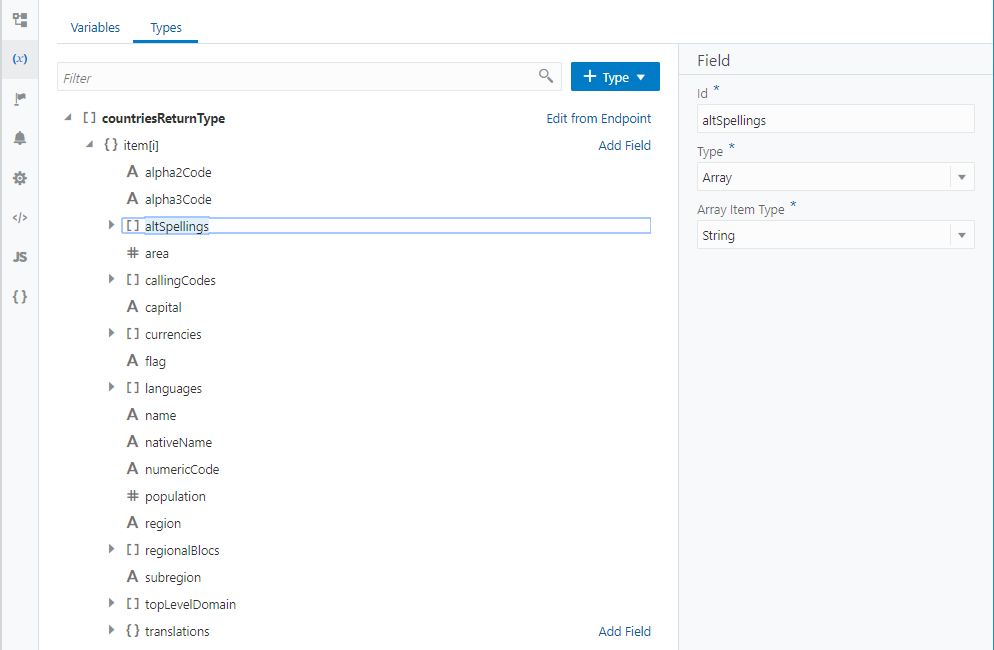
For the LOV we will create a variable at application level that is loaded only once.

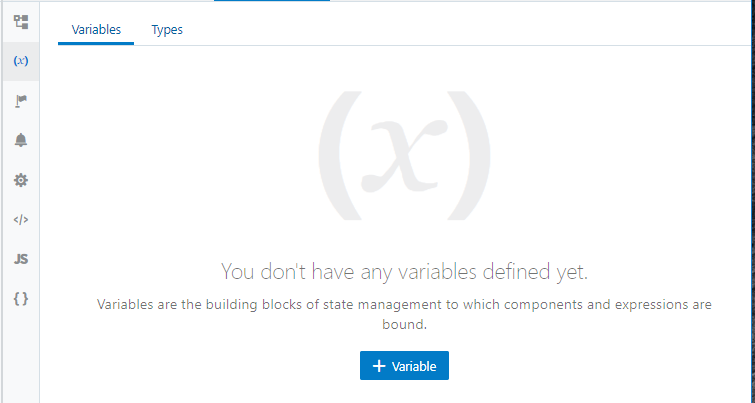


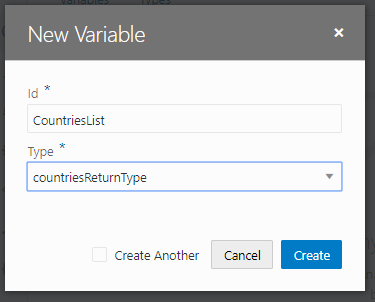
Create a Type from the Countries Endpoint

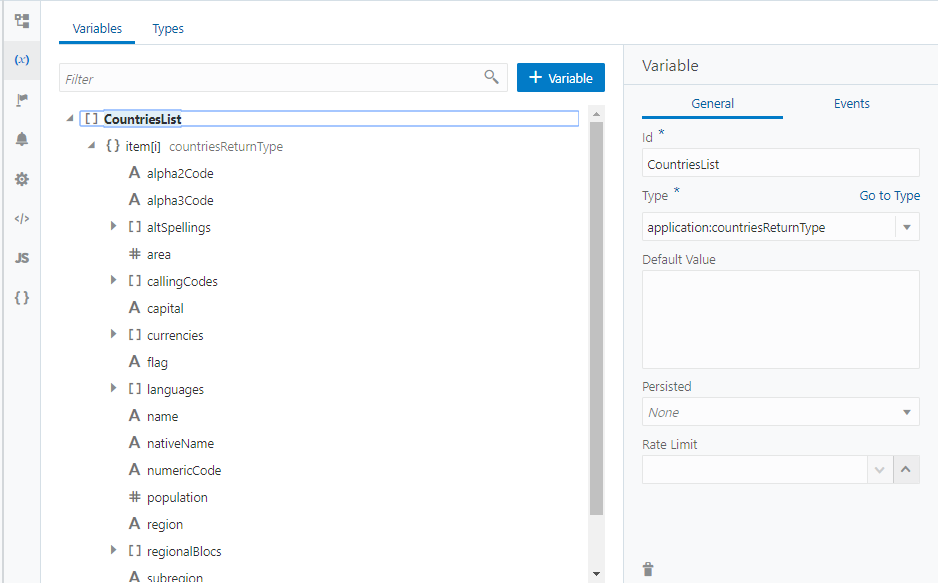








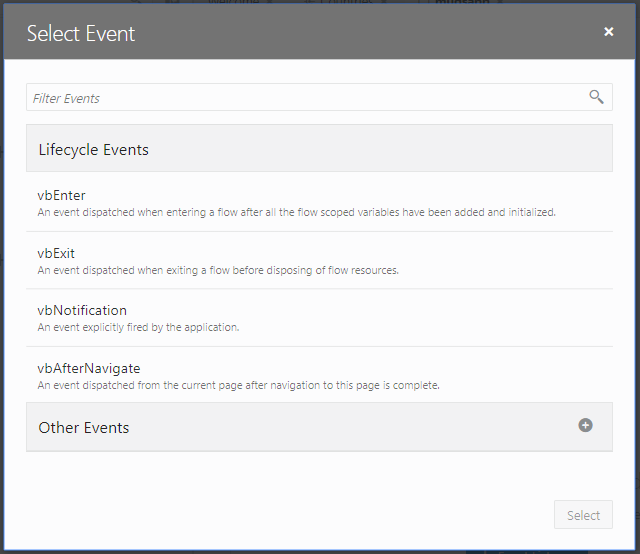


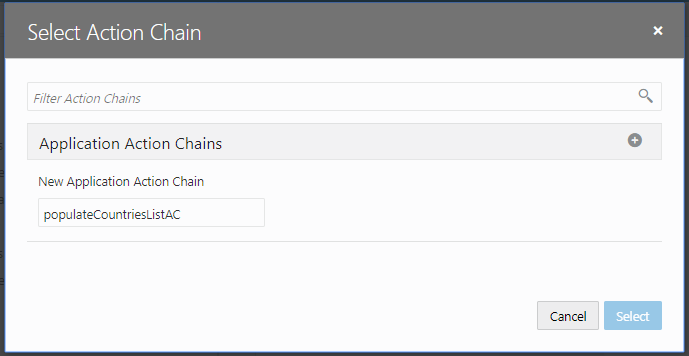


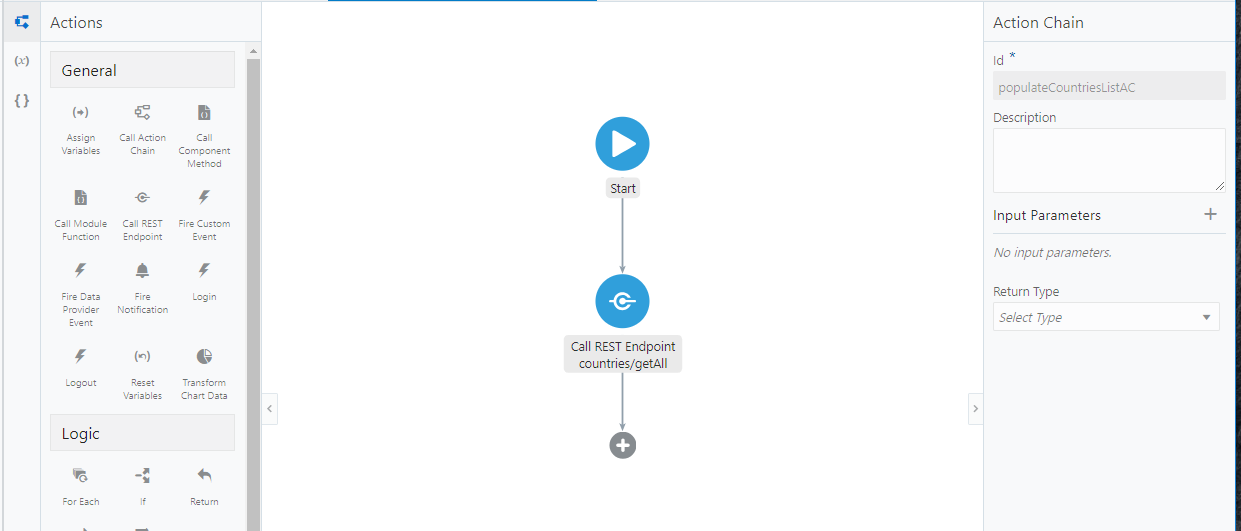
#### Populate CountriesList

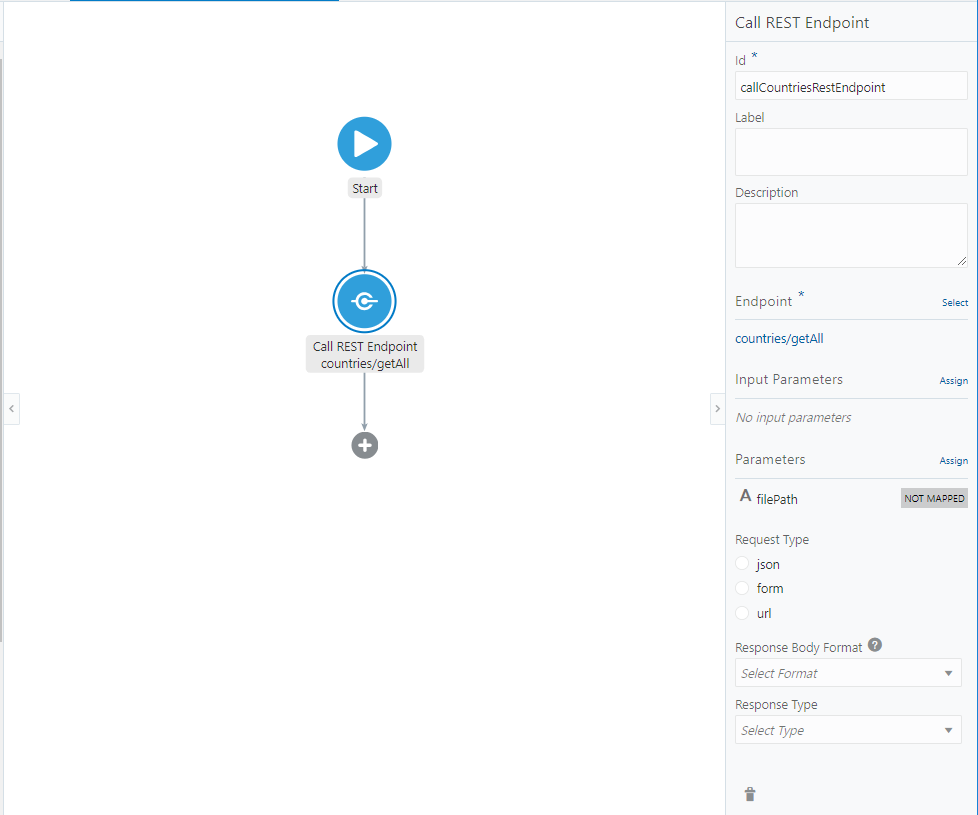
We populate the list by creating a Action Chain that is initiated by an Event Listener for the vbEnter event.

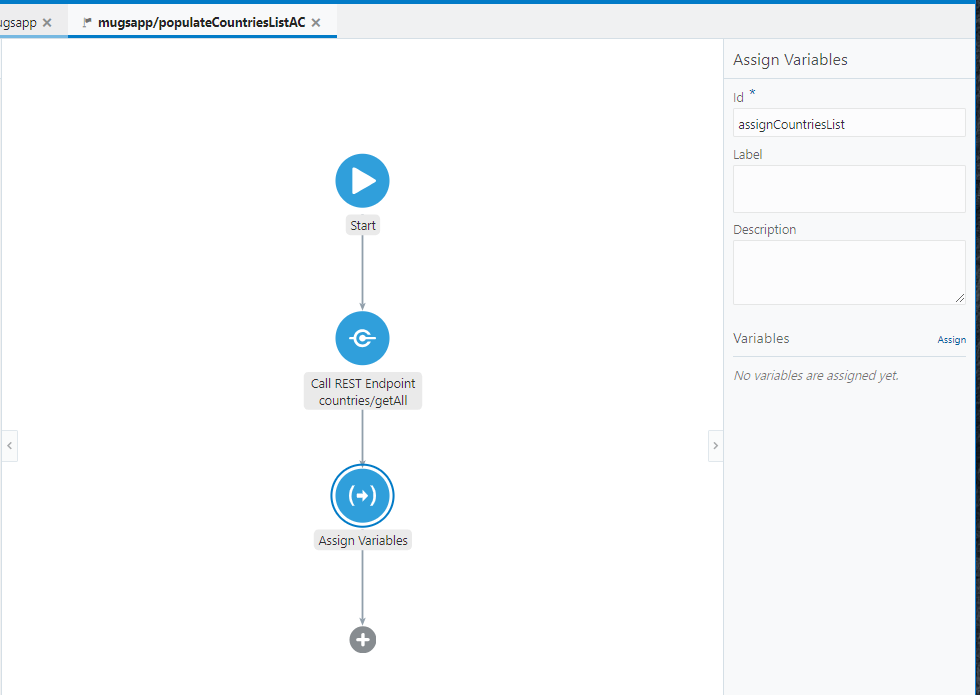


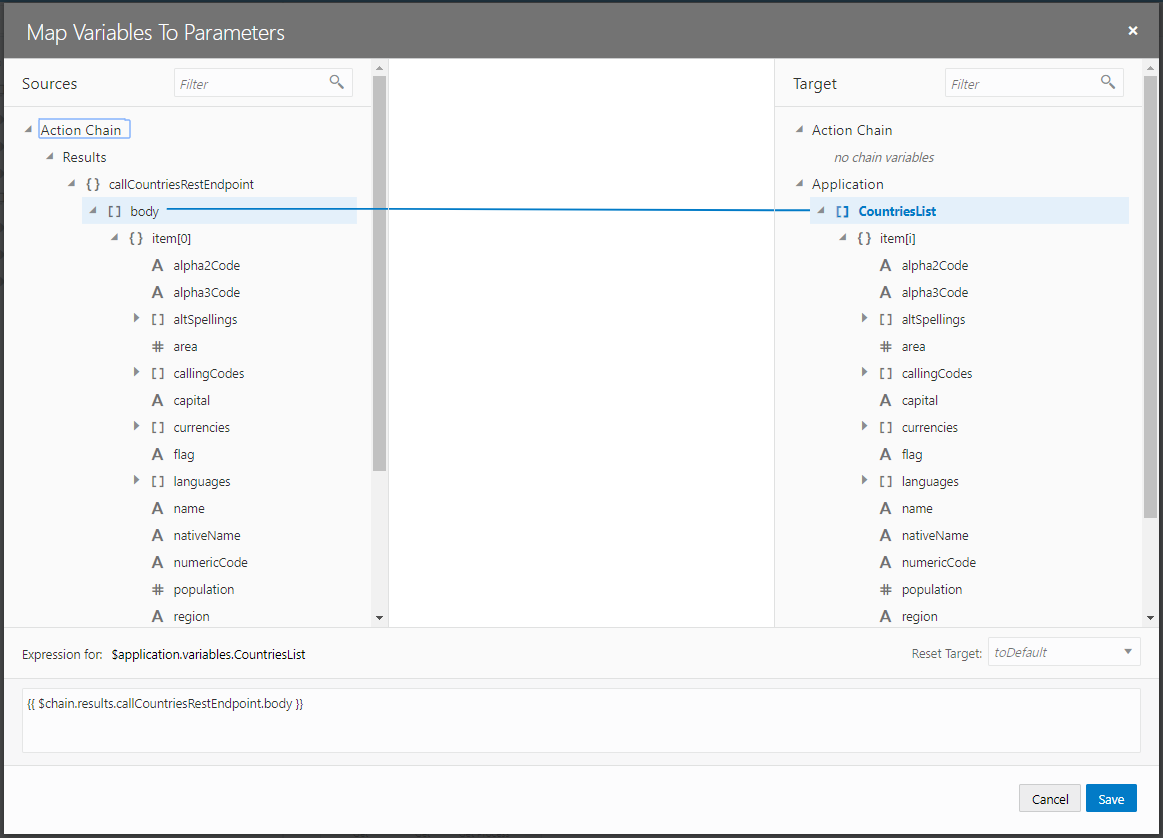






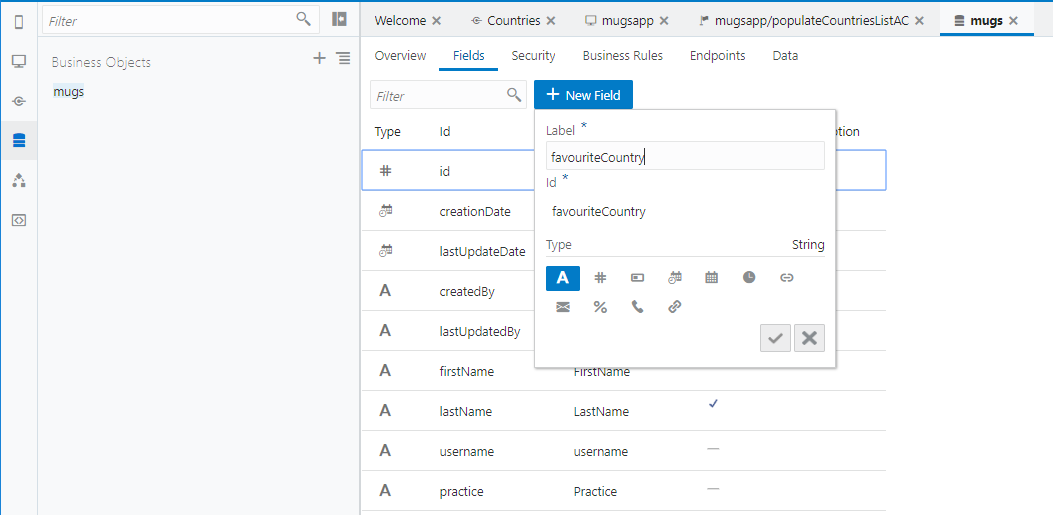




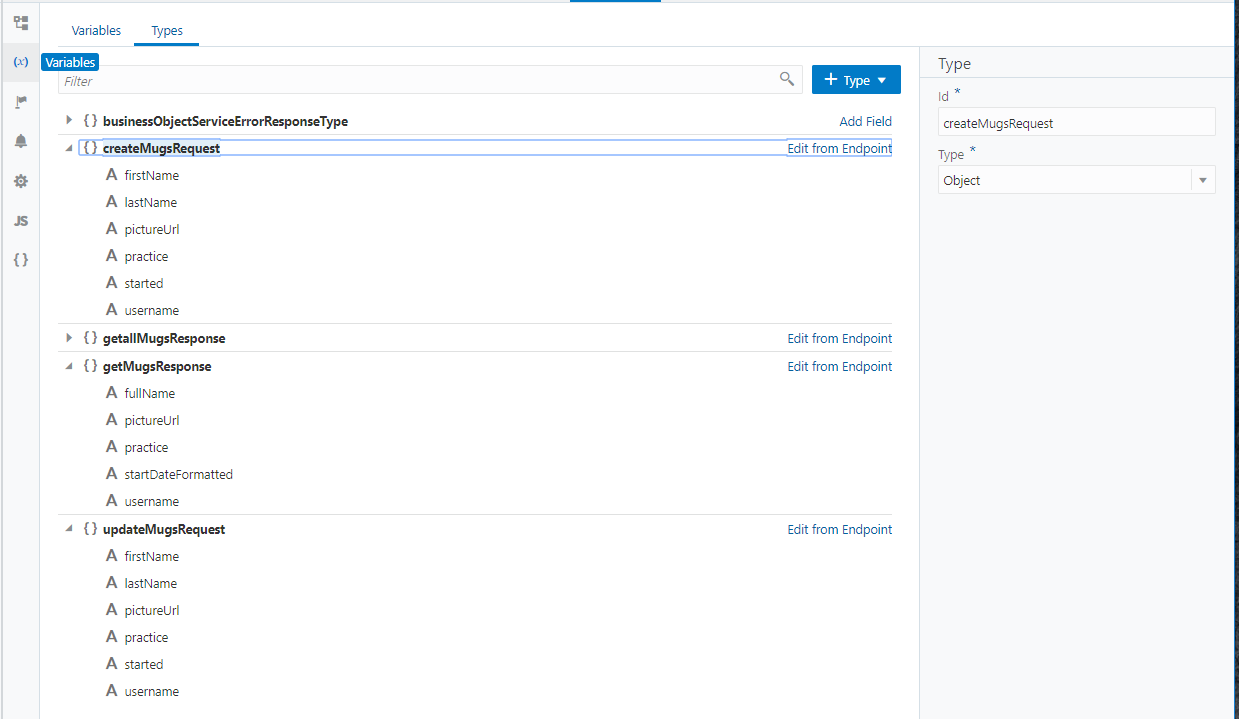


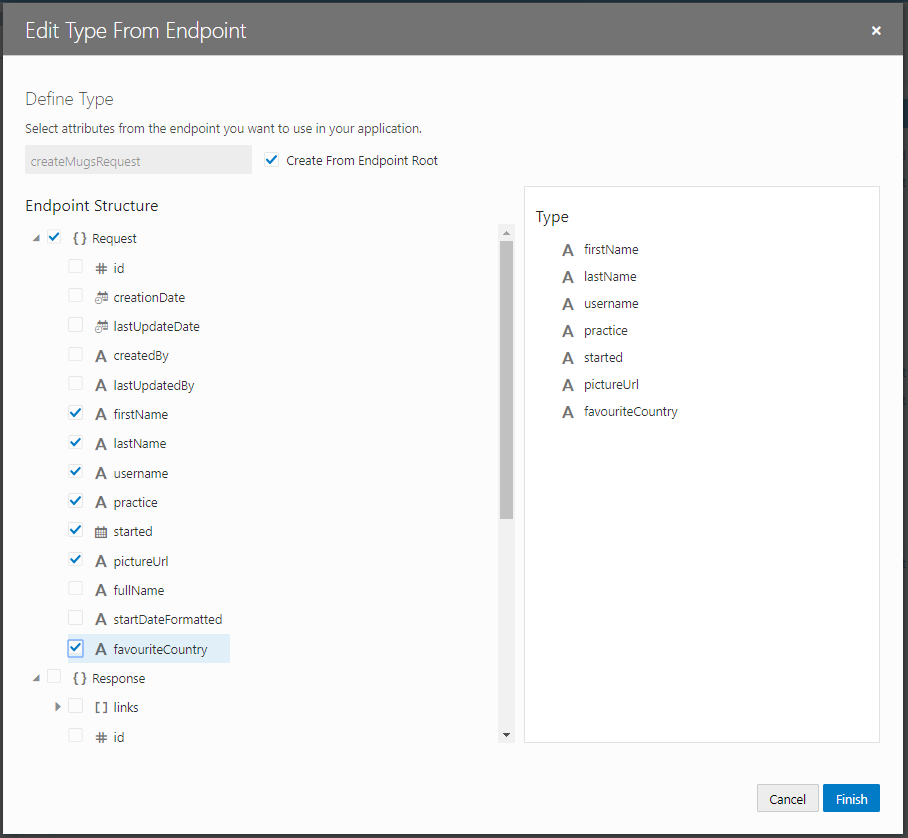
#### Add field

We first have to add the field to the business object.



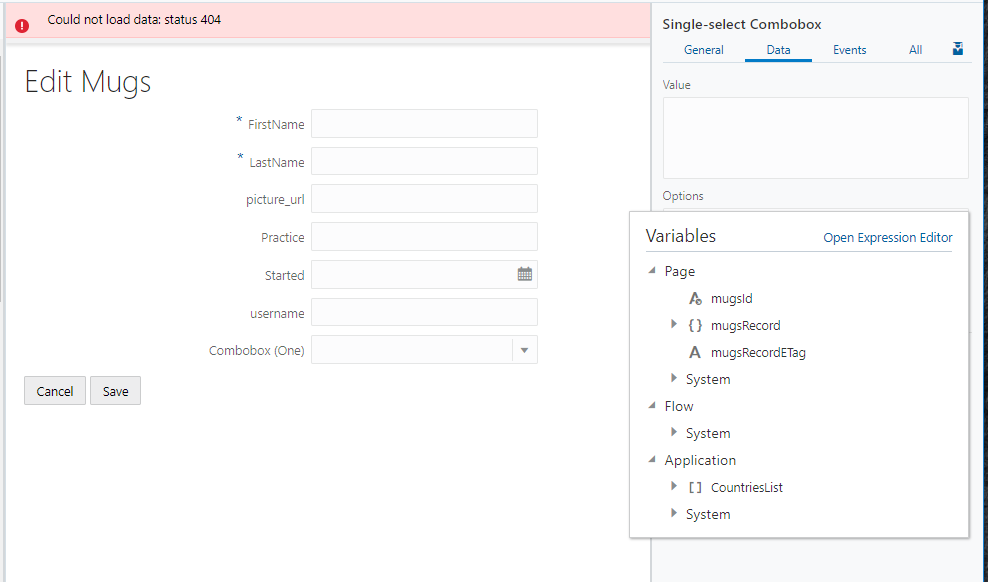
Next we will add this field to 3 types in the main flow.



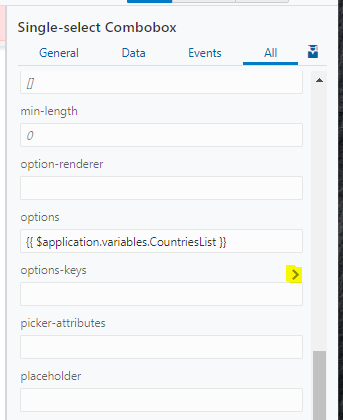


#### Create an LOV

Drop a Combobox One in the Edit page and select the CountriesList for the options.



In the All Tab expand the options-keys attribute





Check if the value is saved! If not, try to find out why.

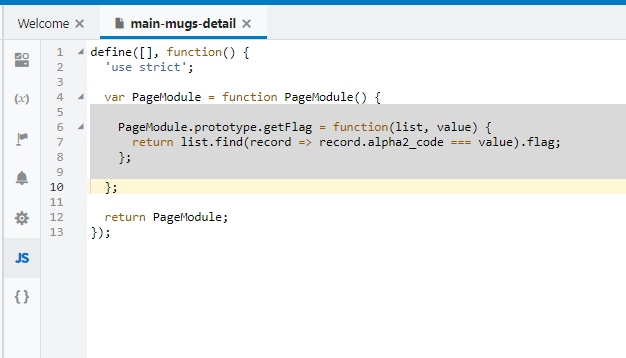
Show flag for favourite country in Detail screen

Add the getFlag function to the DetailPage javascript.

PageModule.prototype.getFlag = function(list, value) {

return list.find(record => record.alpha2Code === value).flag;

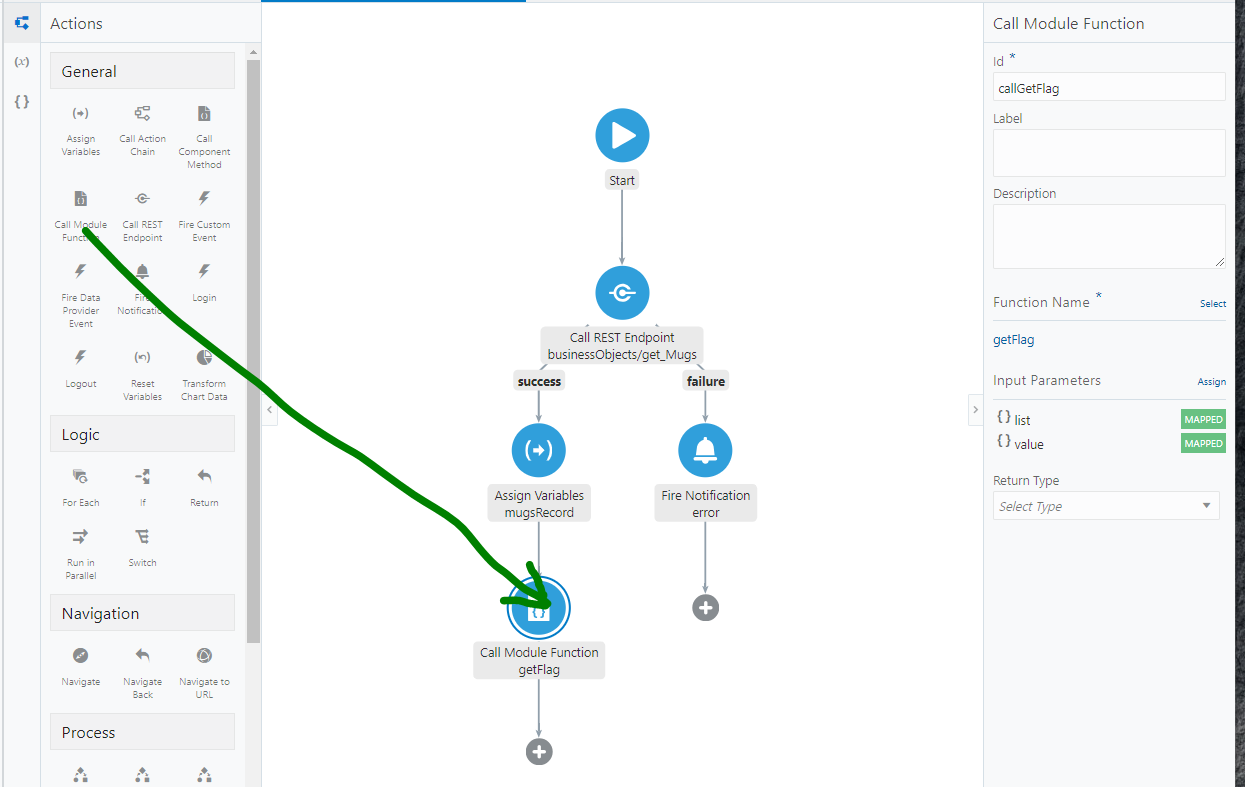
};

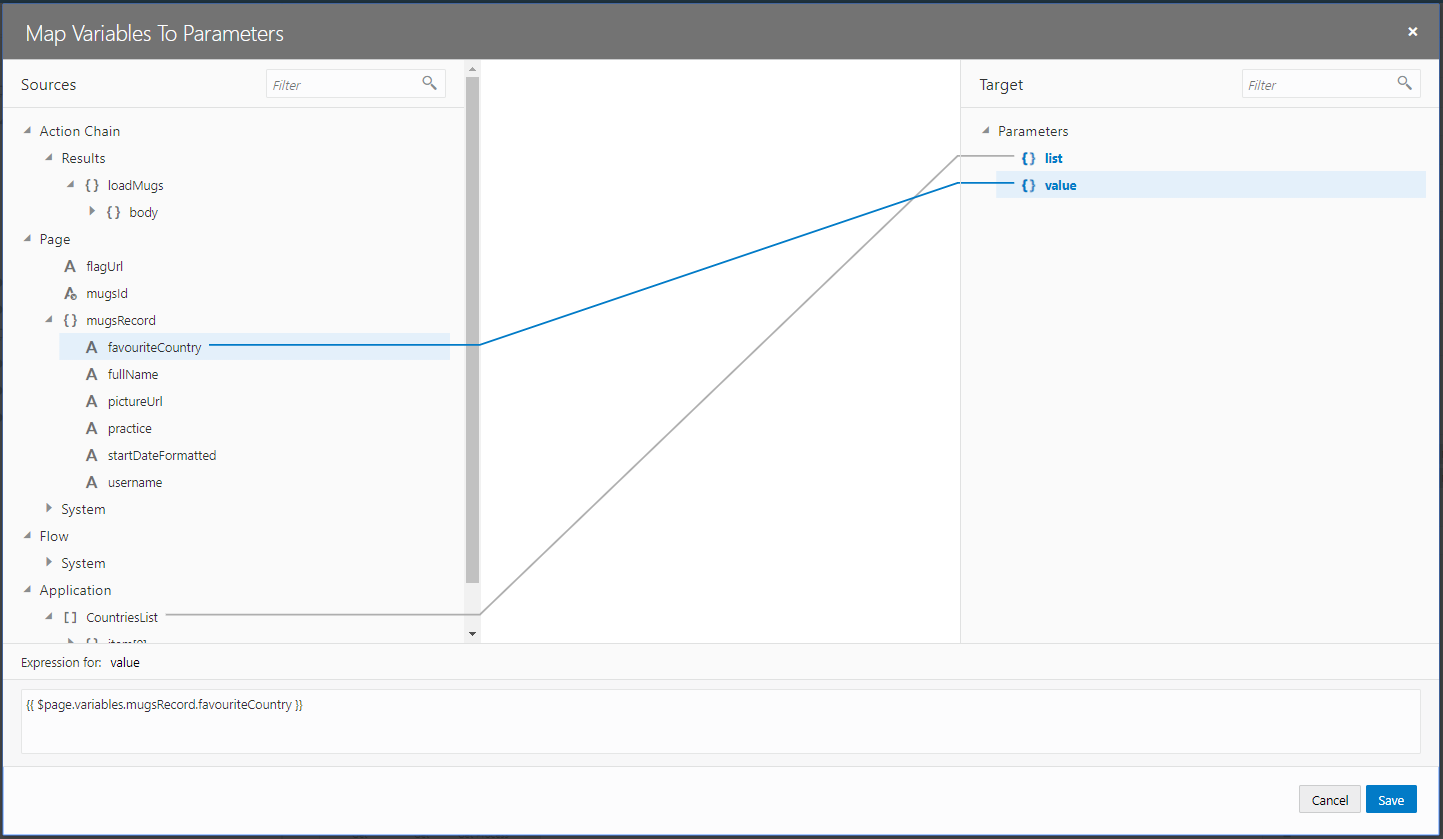


First add a Text Variable named flagUrl.

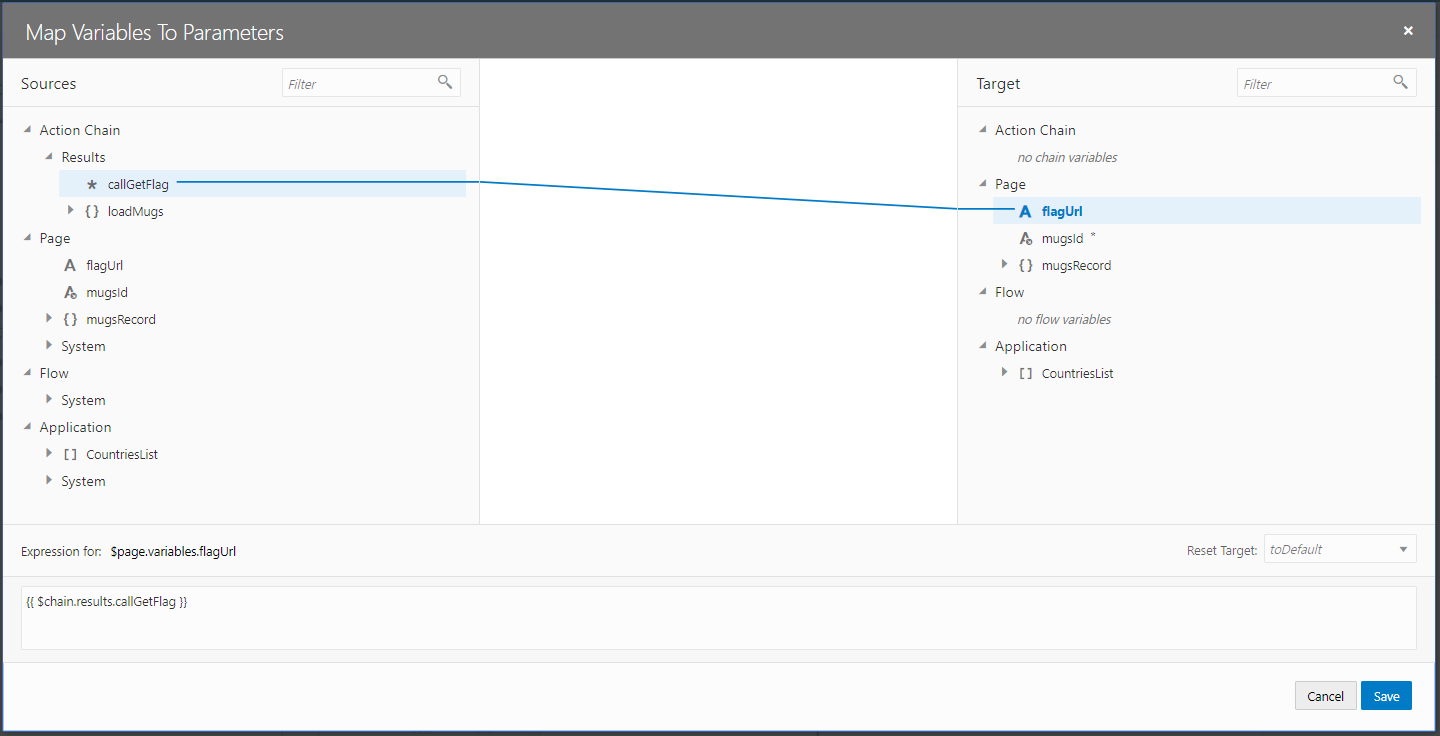
We will populate this variable in the loadMugsChain Action Chain

Add a Call Module Function Action and map the parameters





Map the result to the flagUrl variable

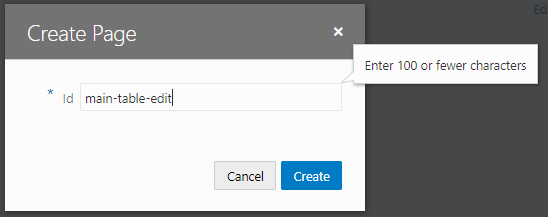


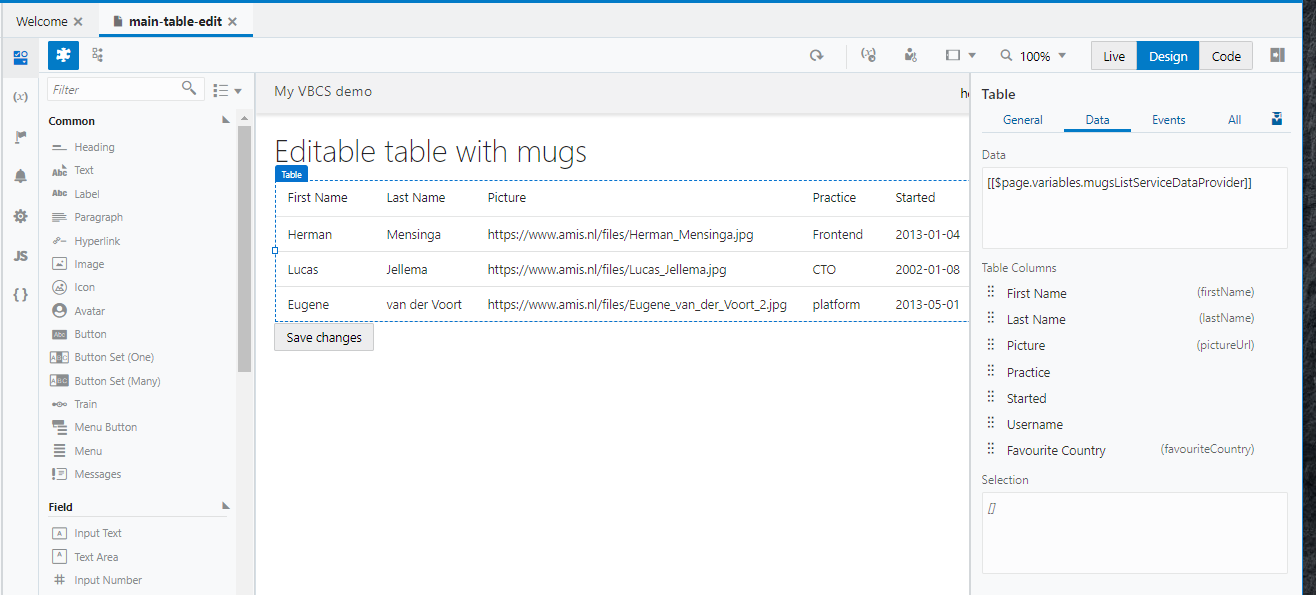
Add this image to the detail page.

Create an editable table

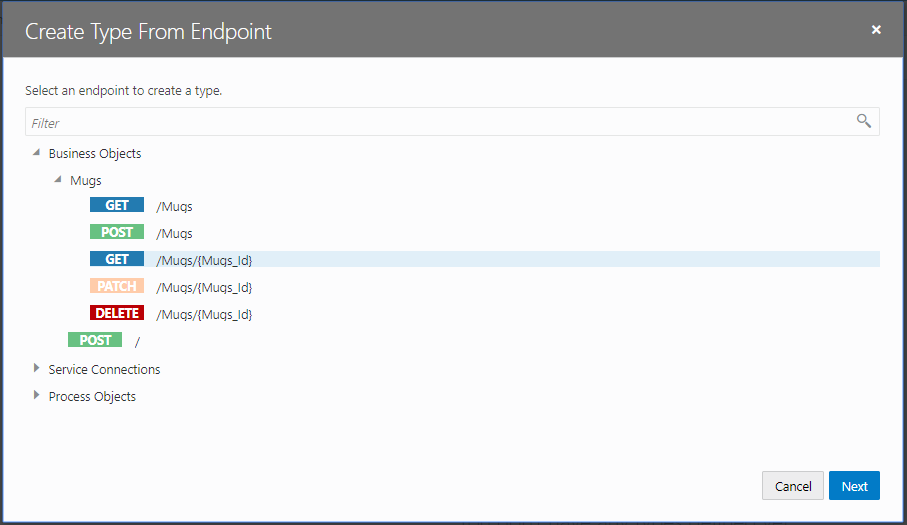
This was inspired by this blog: <https://blogs.oracle.com/vbcs/creating-editable-tables-in-oracle-visual-builder-cloud-service>

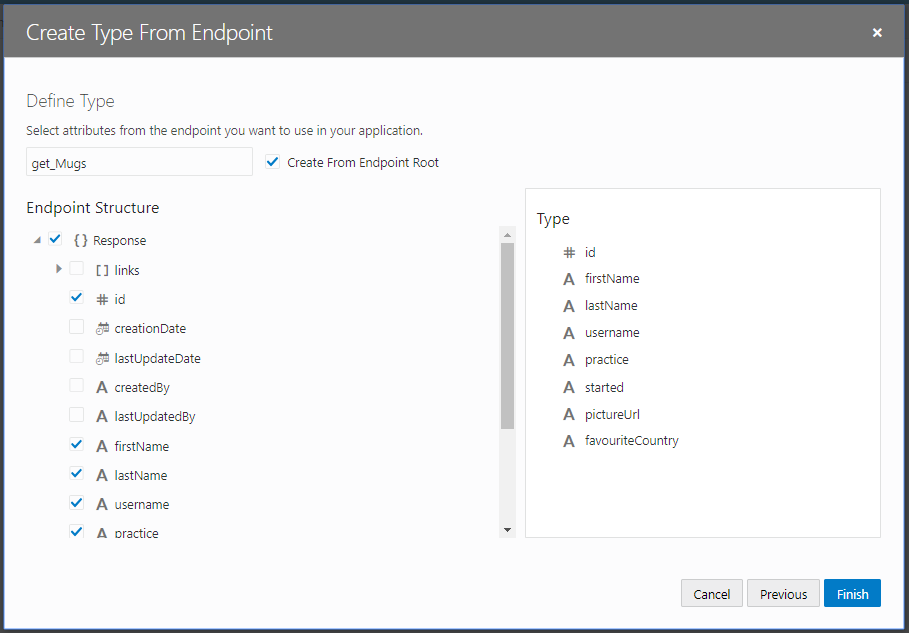
Create a new page

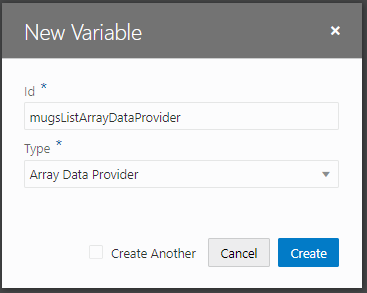


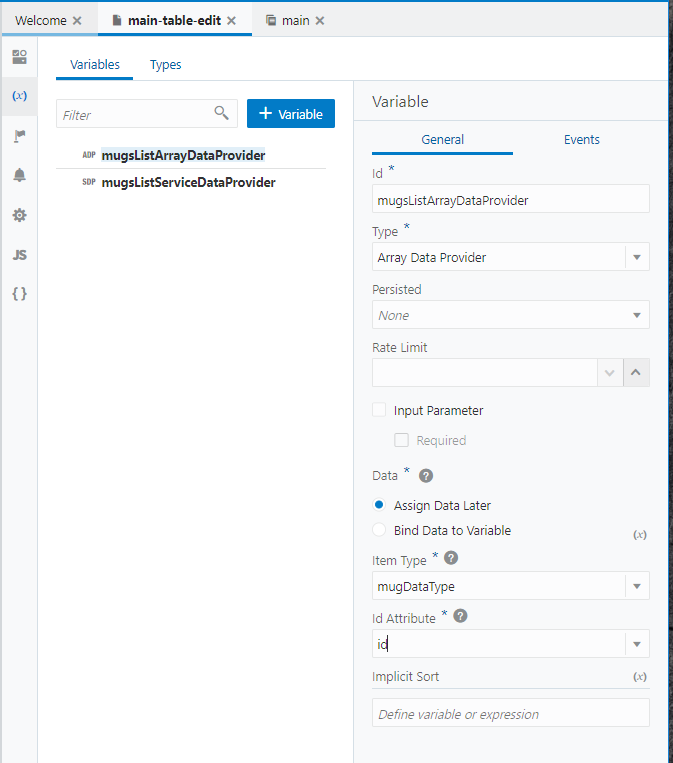


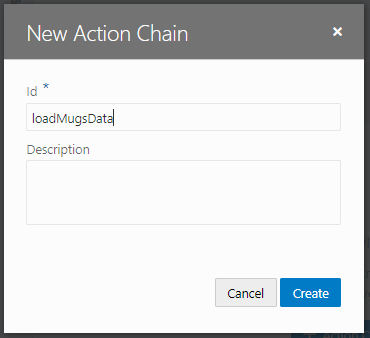
Create Array Data Provider

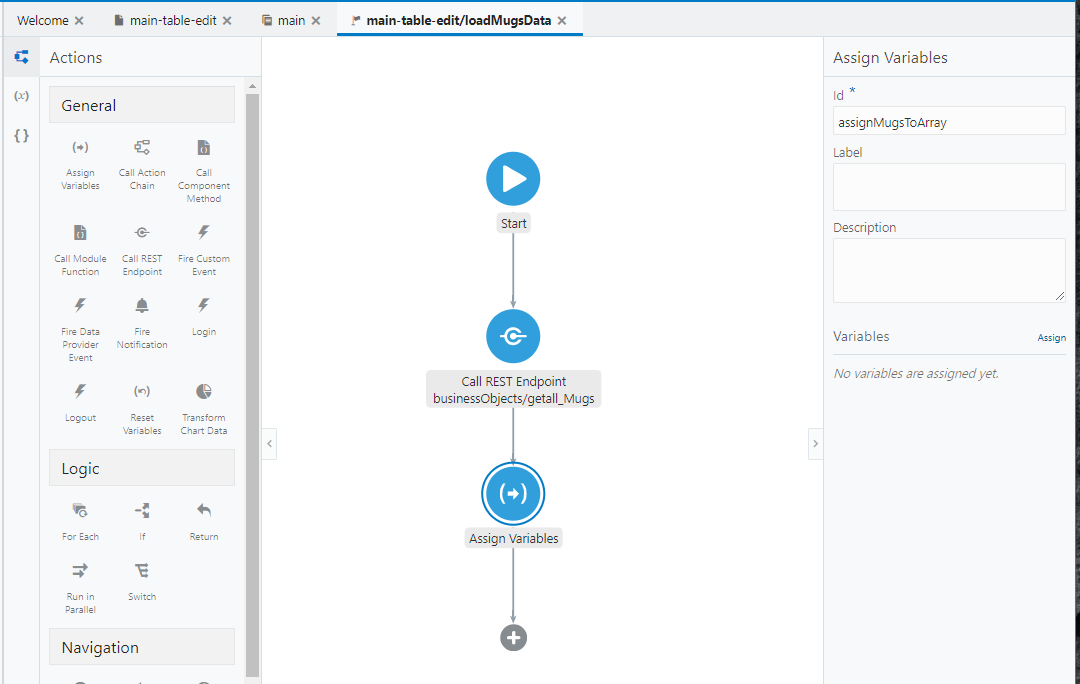


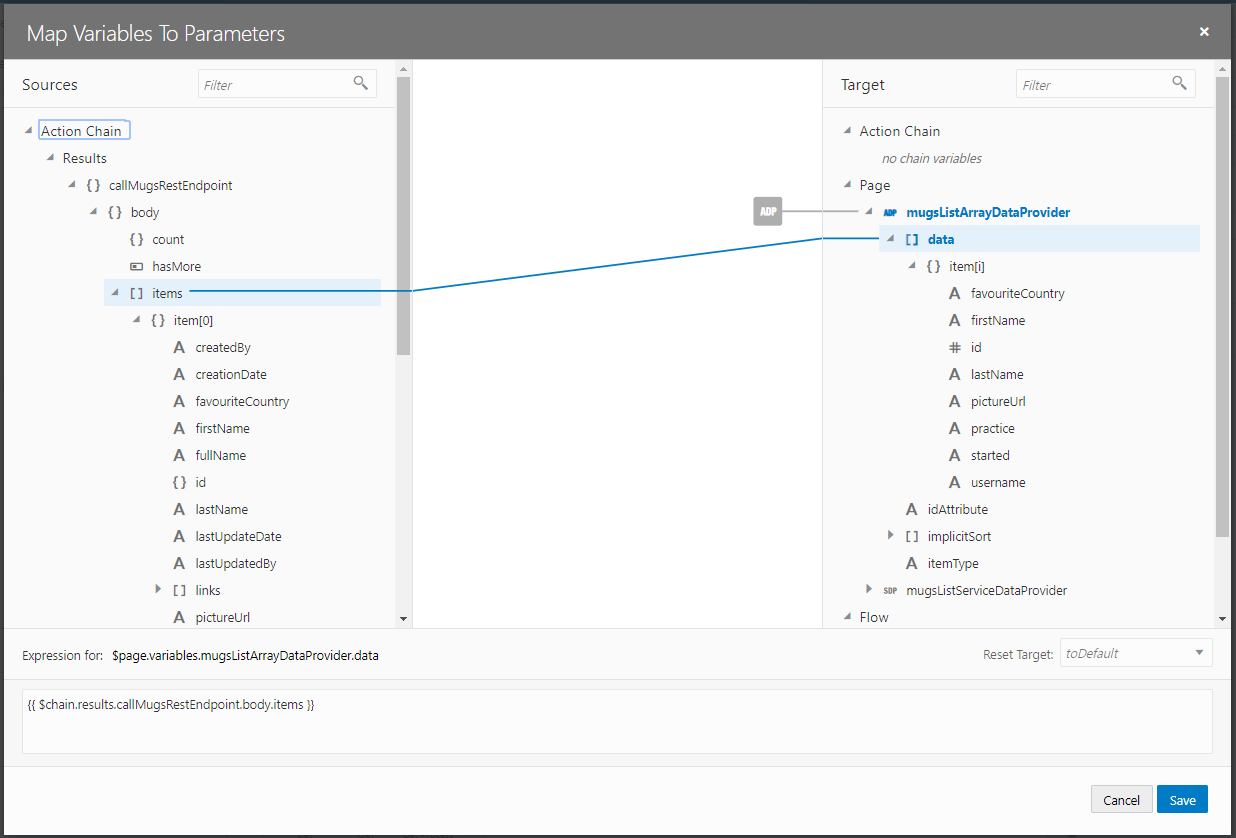


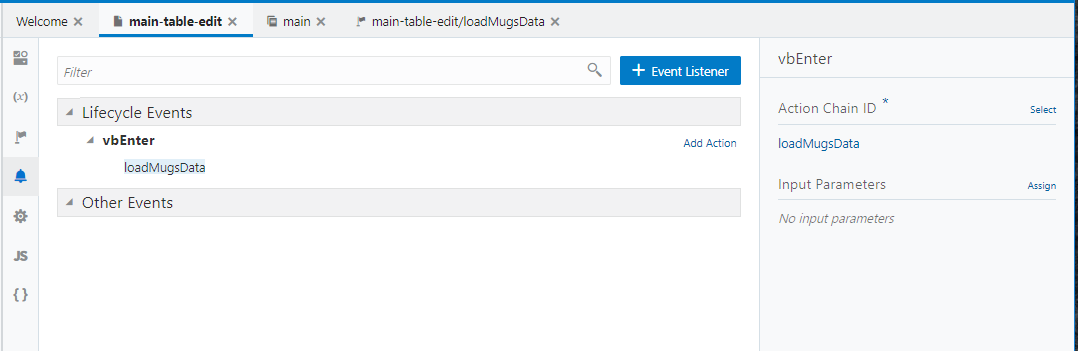




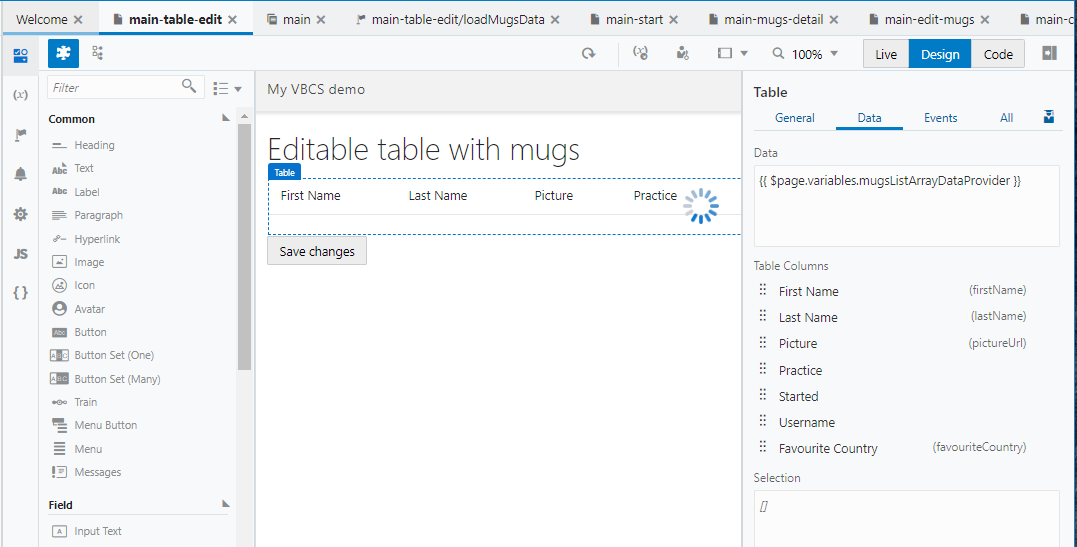




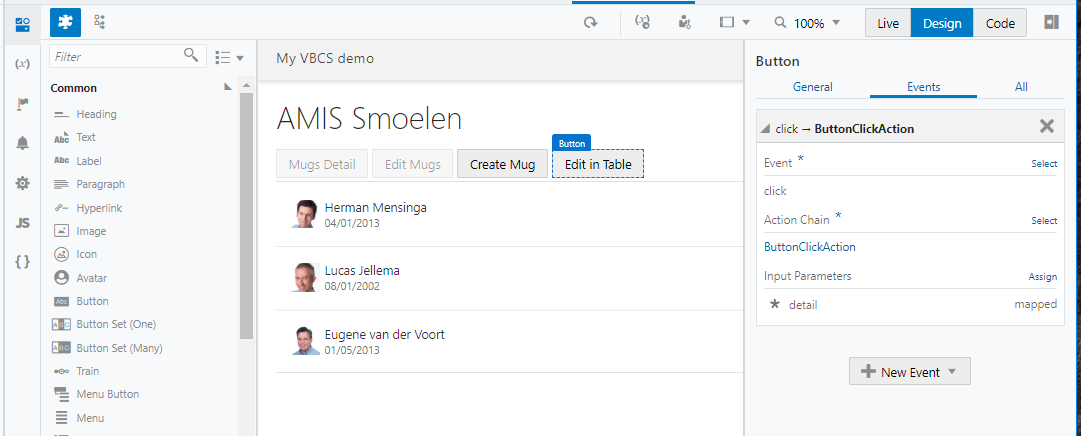




Update the Data Provider



*Add a Button that navigates to the page*



Test if the table now shows data

Set edit-mode to row-edit



In the HTML-code add the two RowTemplates

<script type="text/html" id="rowTemplate">

<tr>

<td data-bind="text: firstName”></td>

<td data-bind="text: lastName”></td>

<td data-bind="text: pictureUrl”></td>

<td data-bind="text: practice”></td>

<td data-bind="text: started”></td>

<td data-bind="text: username”></td>

<td data-bind="text: favouriteCountry”></td>

<td data-bind="text: firstName”></td>

</tr>

</script>

<script type="text/html" id="editRowTemplate">

<tr>

<td>

<oj-input-text id="it1" value="{{firstName}}" data-oj-context></oj-input-text>

</td>

<td>

<oj-input-text id="it2" value="{{lastName}}" data-oj-context>

</oj-input-text>

</td>

<td>

<oj-input-text id="it3" value="{{pictureUrl}}" data-oj-context>

</oj-input-text>

</td>

<td>

<oj-input-text id="it4" value="{{practice}}" data-oj-context>

</oj-input-text>

</td>

<td>

<oj-input-text id="it5" value="{{started}}" data-oj-context>

</oj-input-text>

</td>

<td>

<oj-input-text id="it6" value="{{username}}" data-oj-context>

</oj-input-text>

</td>

<td>

<oj-input-text id="it7" value="{{favouriteCountry}}"

data-oj-context></oj-input-text>

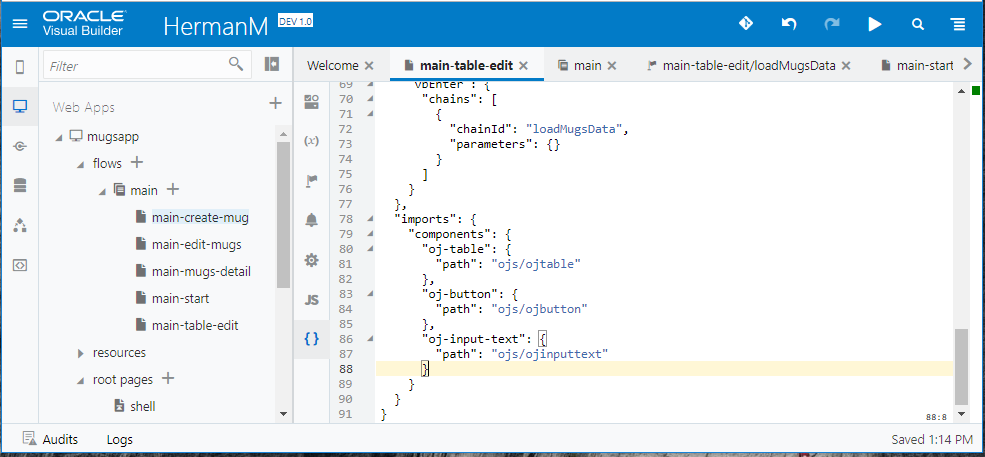
</td>

</tr>

</script>

Eg. In the div containing the oj-table.

Be sure that the inputtext component is imported



Next we create a function to select the right rowTemplate

define(['ojs/ojcore', 'knockout', 'jquery'], function(oj, ko, $) {

'use strict';

var PageModule = function PageModule() {

var self = this;

// function to determine which renderer to use for

// rendering depending on mode

self.\_editRowRenderer = oj.KnockoutTemplateUtils.getRenderer(

'editRowTemplate', true);

self.\_navRowRenderer = oj.KnockoutTemplateUtils.getRenderer(

'rowTemplate', true);

PageModule.prototype.rowRenderer = function(context) {

var mode = context['rowContext']['mode'];

var renderer;

if (mode === 'edit') {

self.\_editRowRenderer(context);

} else if (mode === 'navigation') {

self.\_navRowRenderer(context);

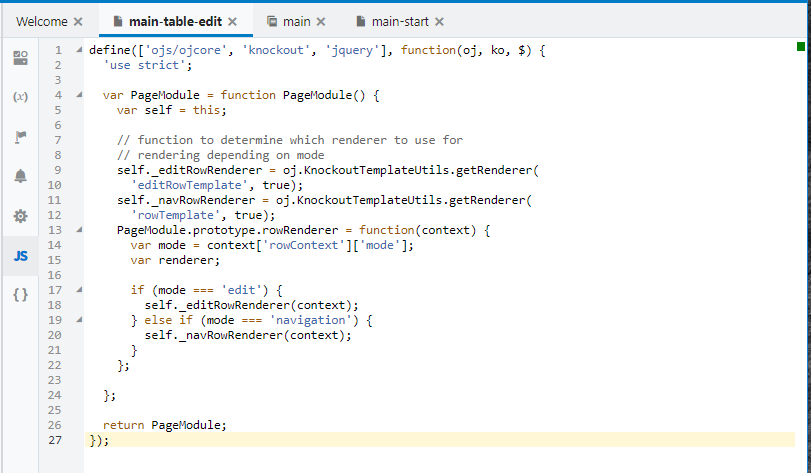
}

};

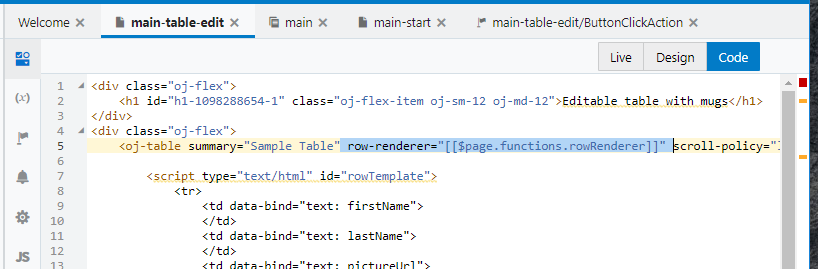
};

return PageModule;

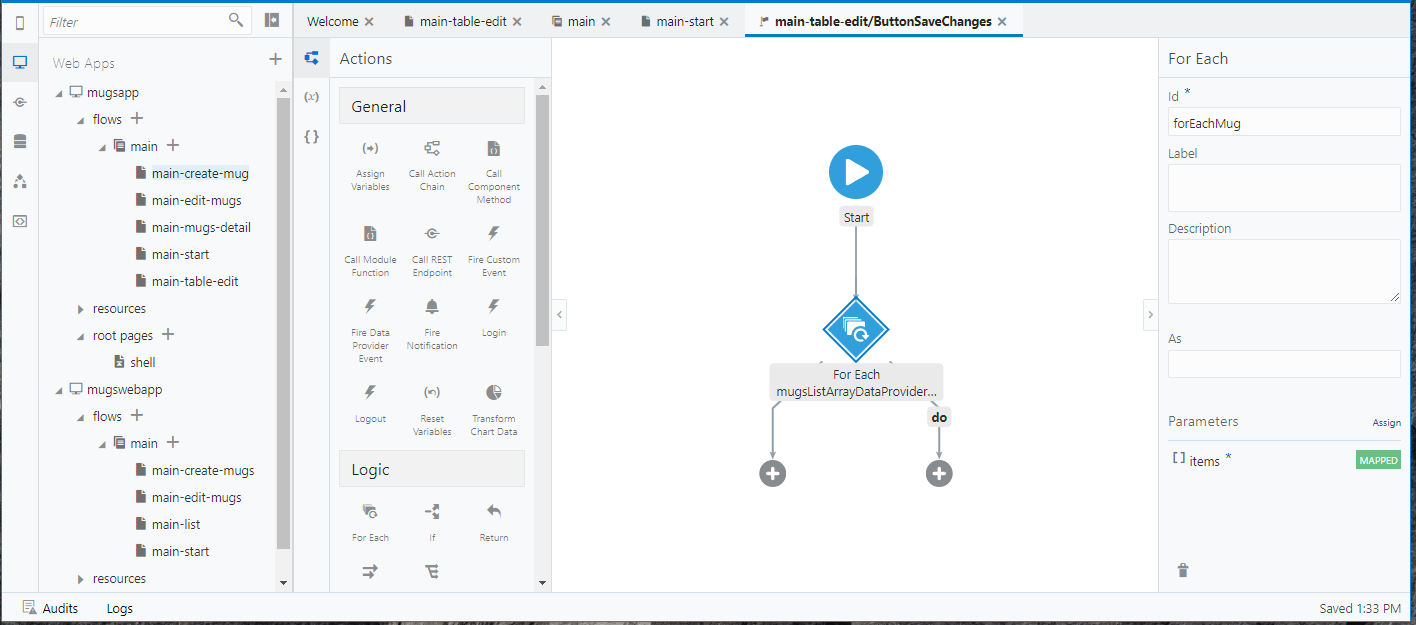
});

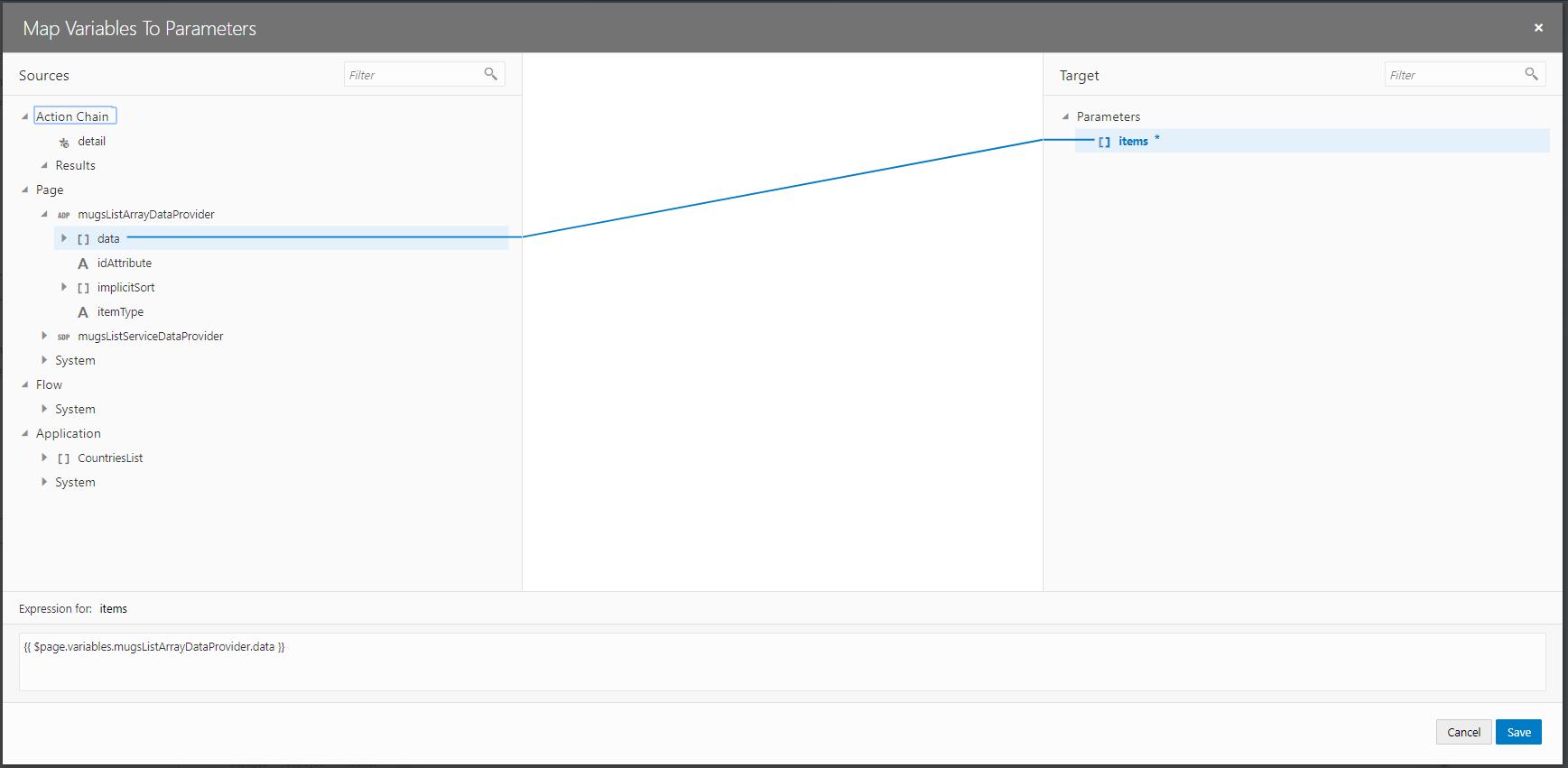


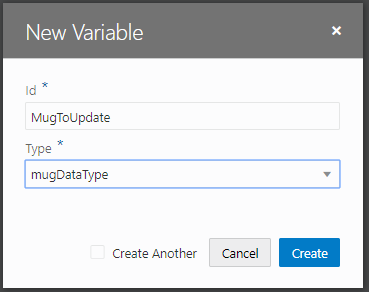
Set the rowRenderer using the created function:

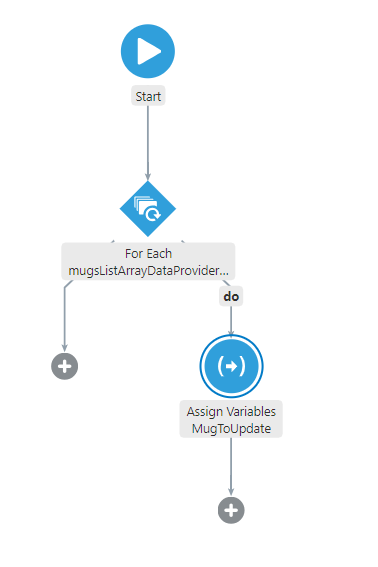


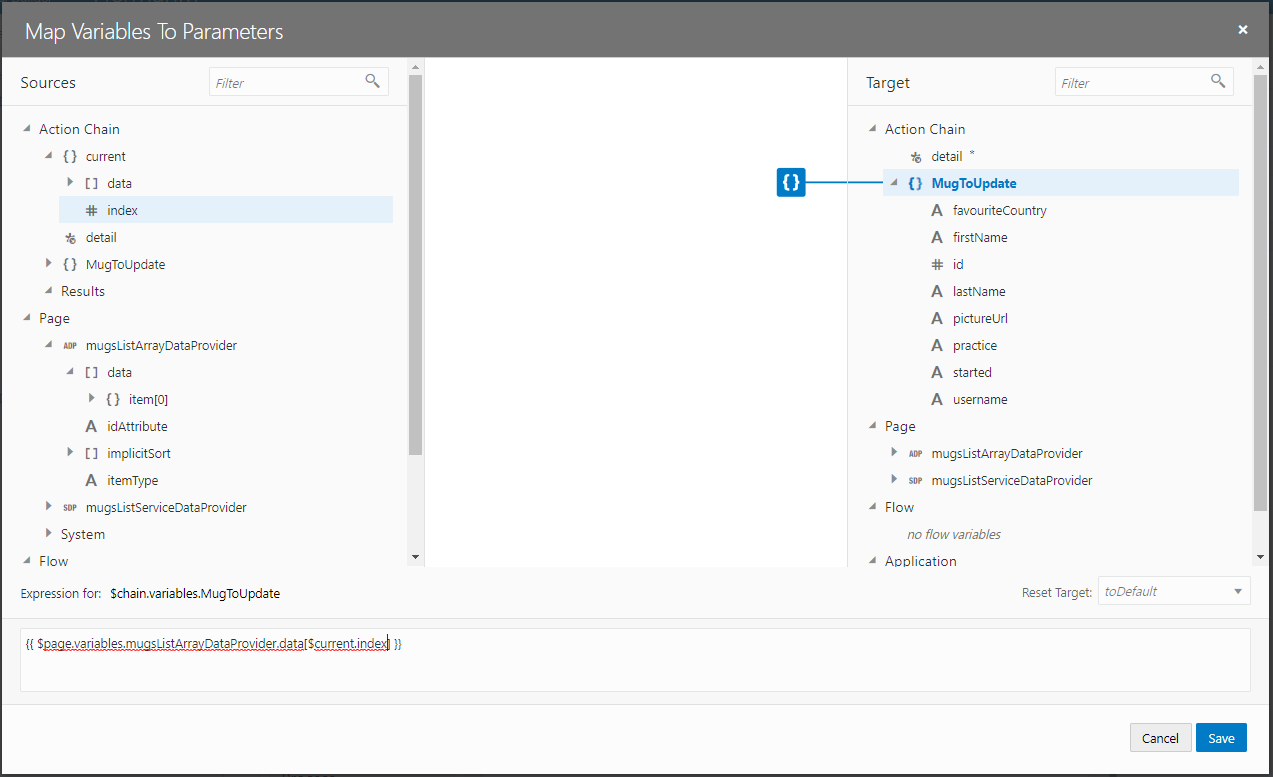
Next we will Save the Data in a For-Each loop initiated from a Button.



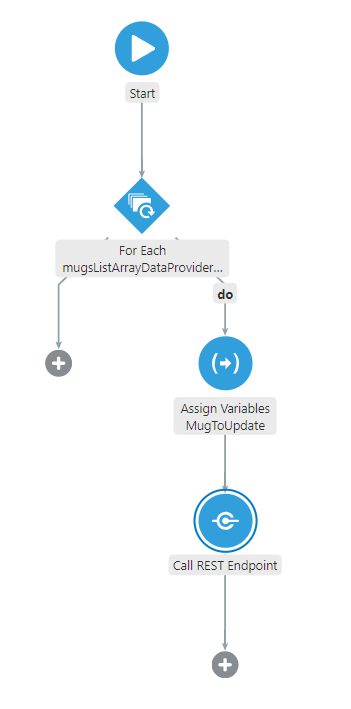


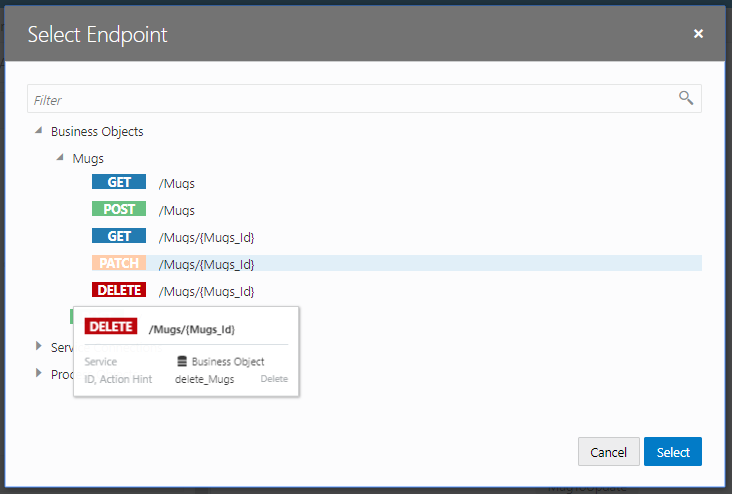


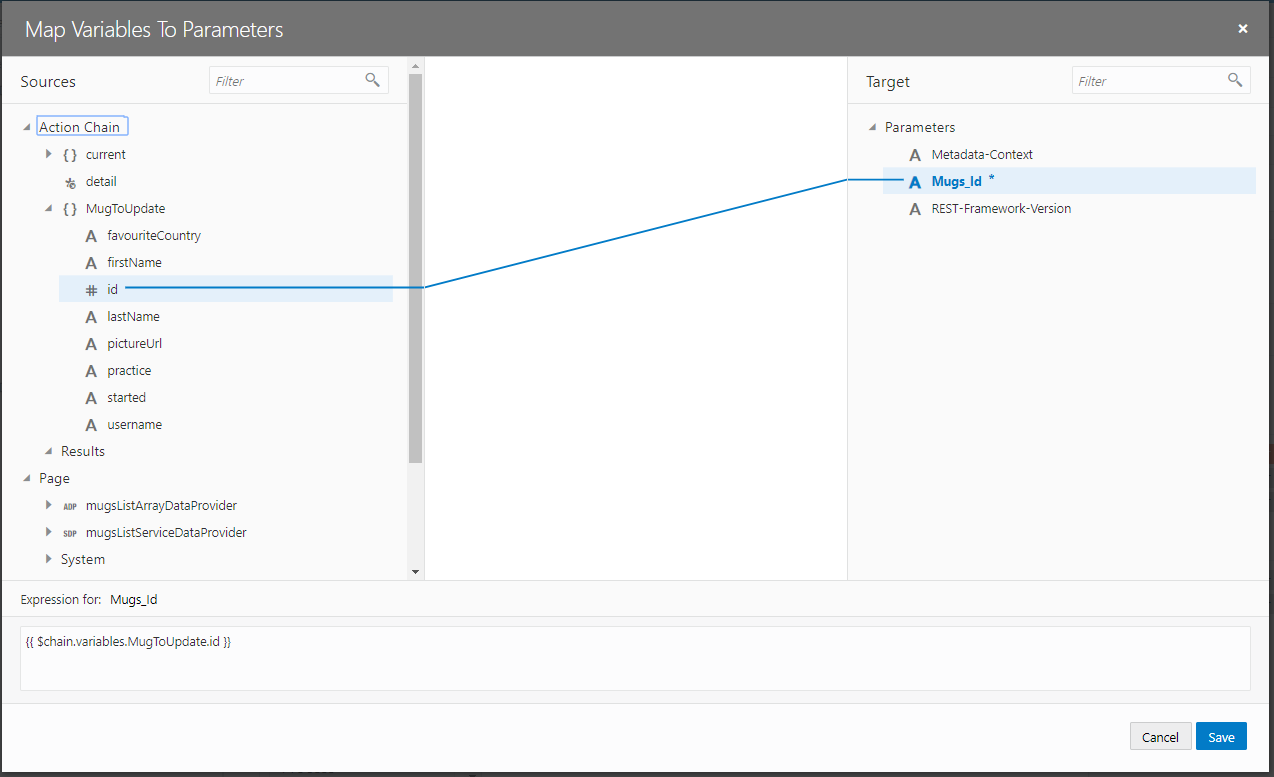


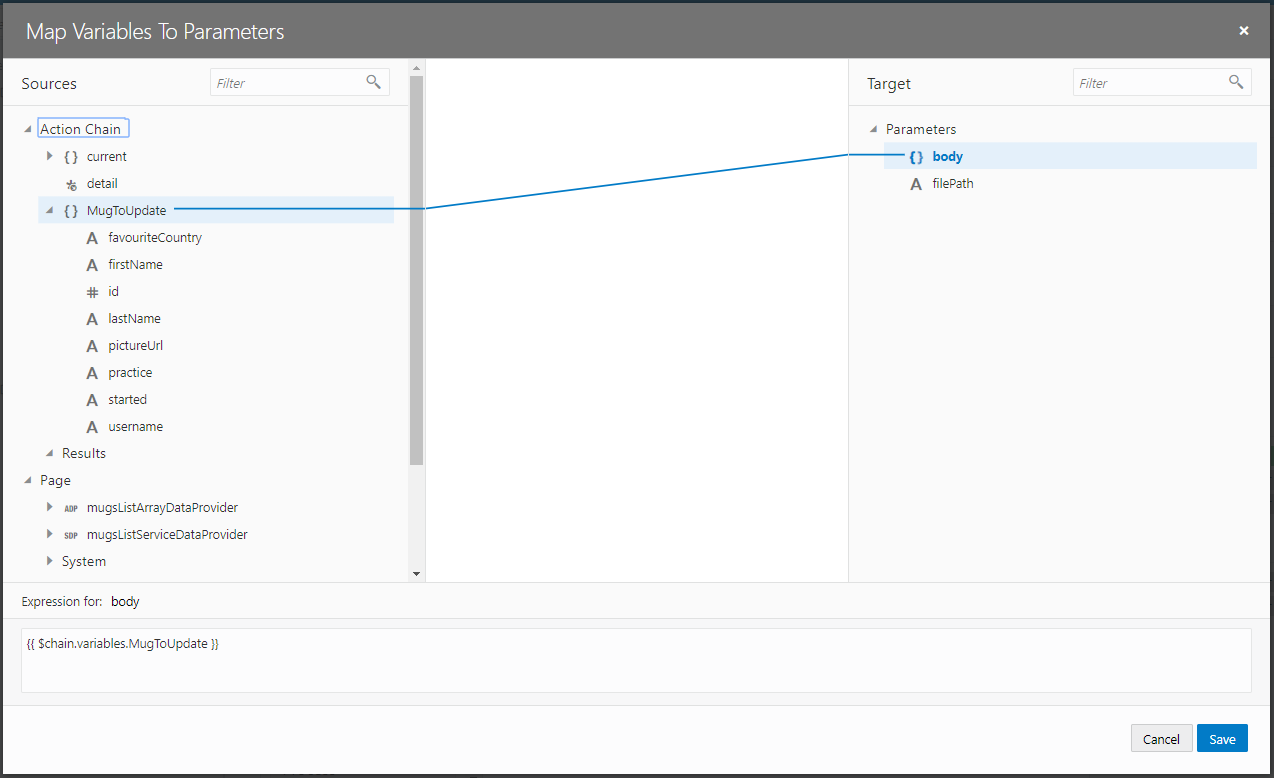


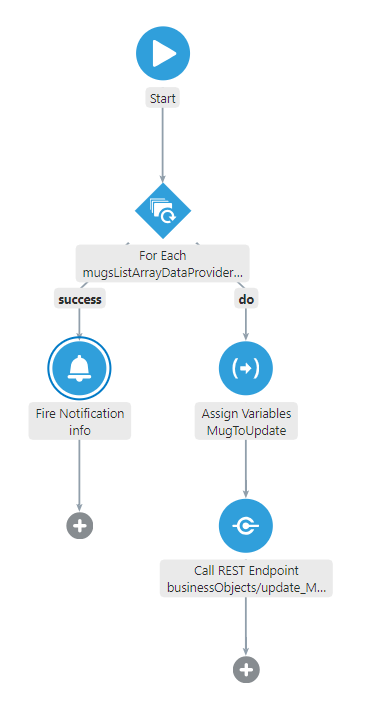
Assign {{ $page.variables.mugsListArrayDataProvider.data[$current.index] }} to MugToUpdate variable

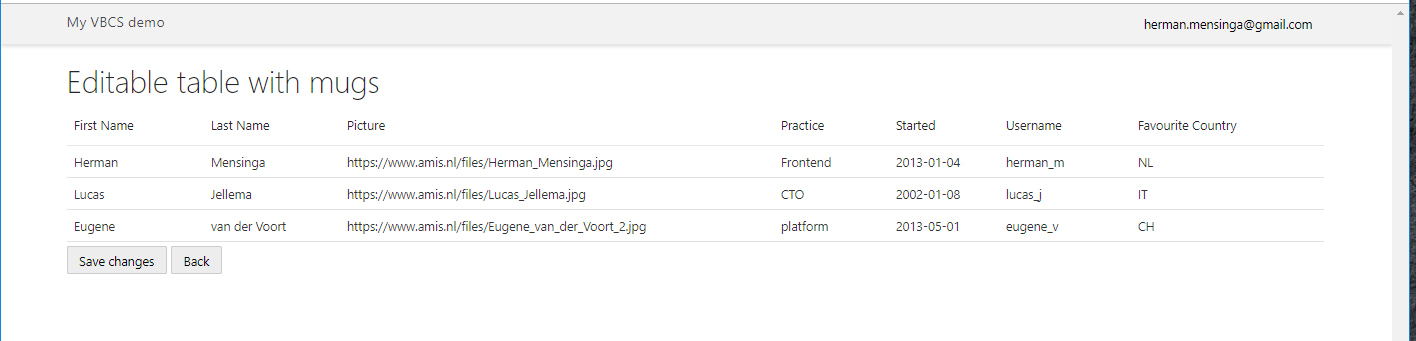








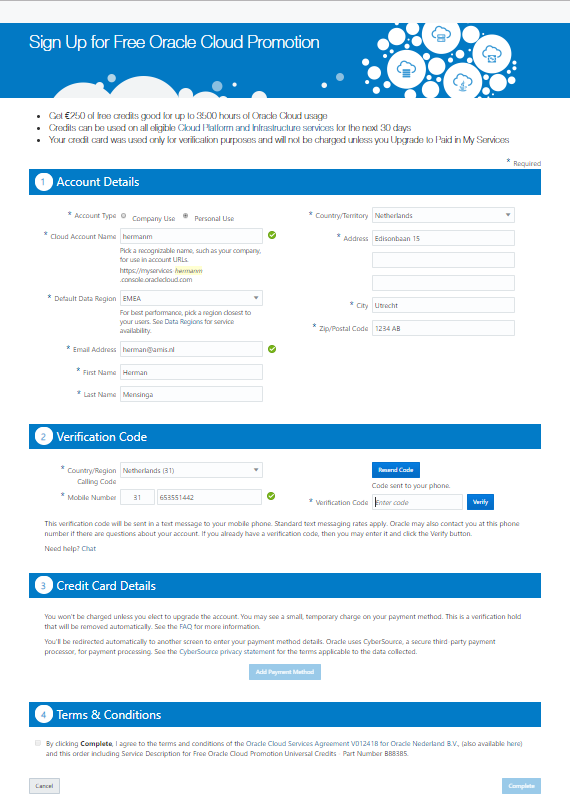


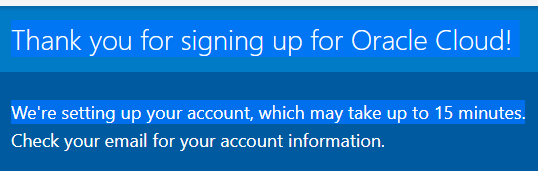


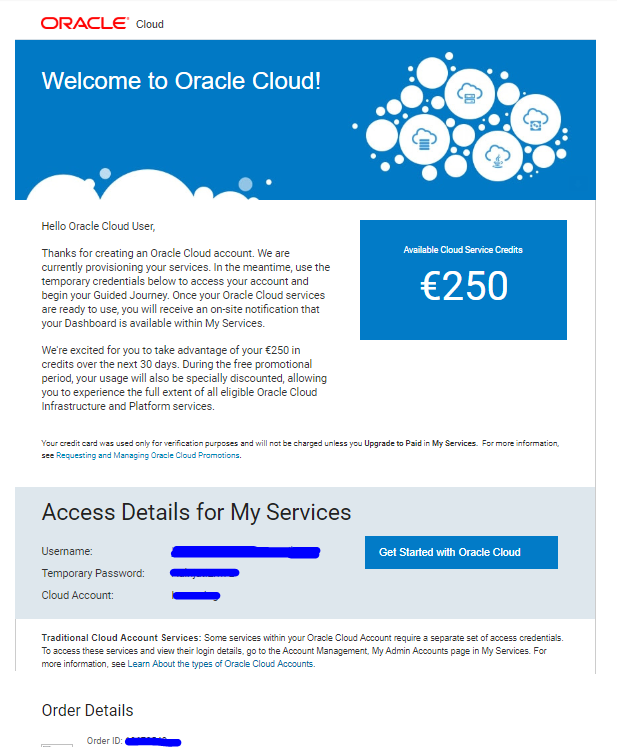
You could change the Favourite Country column in a Combobox or the Started column in a inputDate.

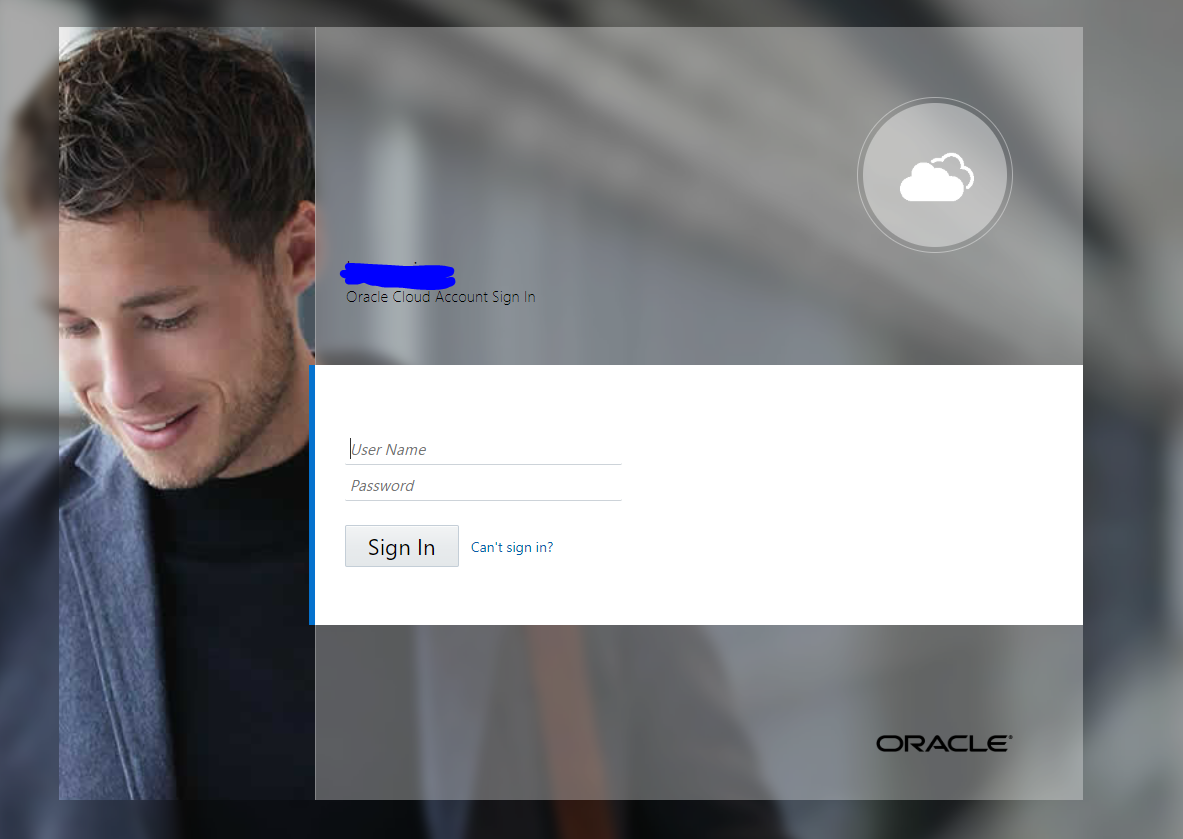
## Sign up for a trial account

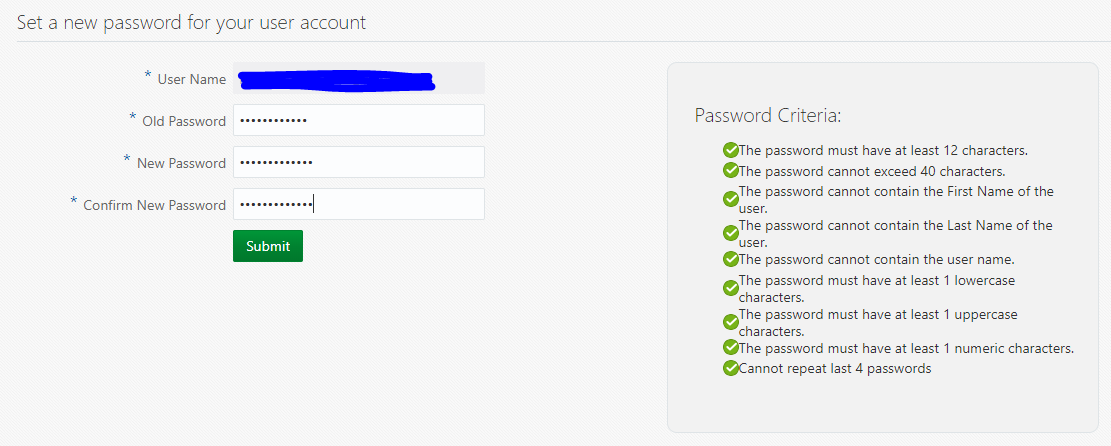
<https://myservices.us.oraclecloud.com/mycloud/signup>

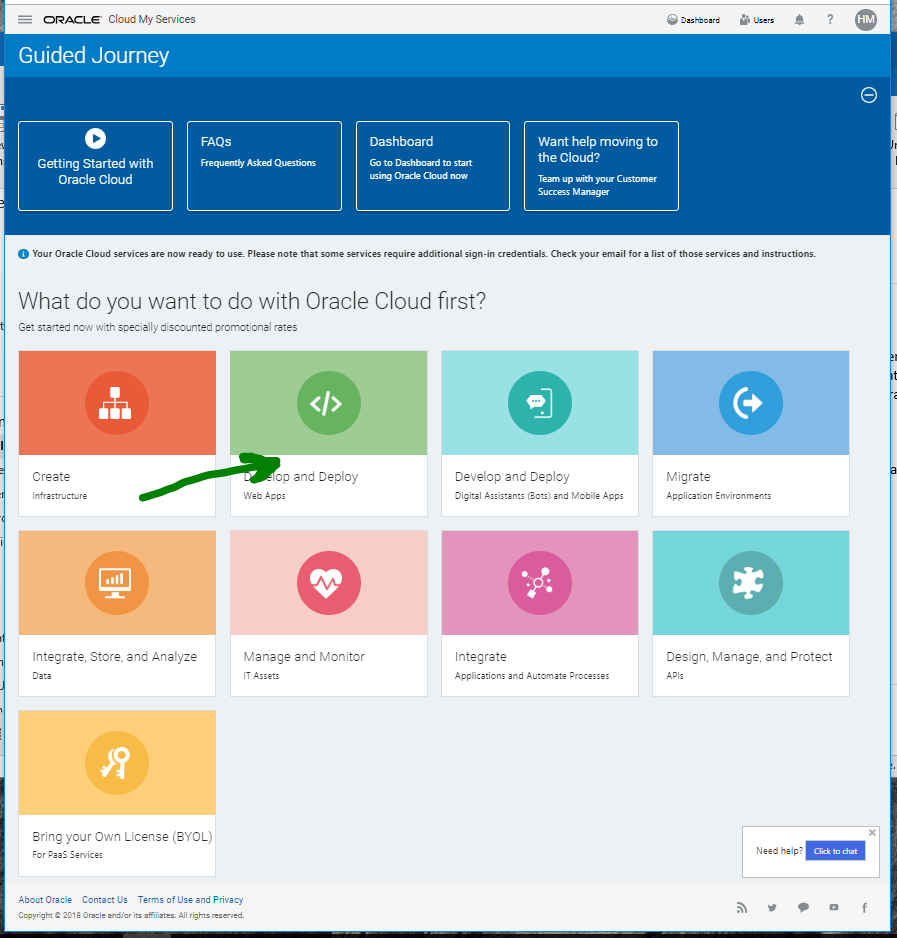


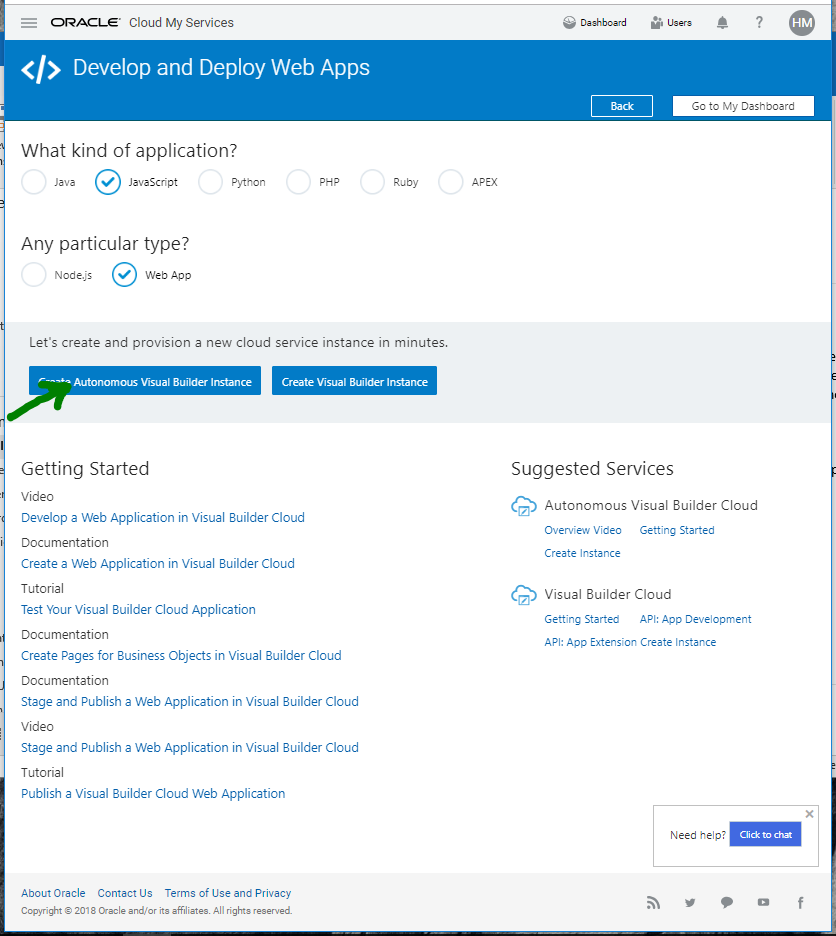


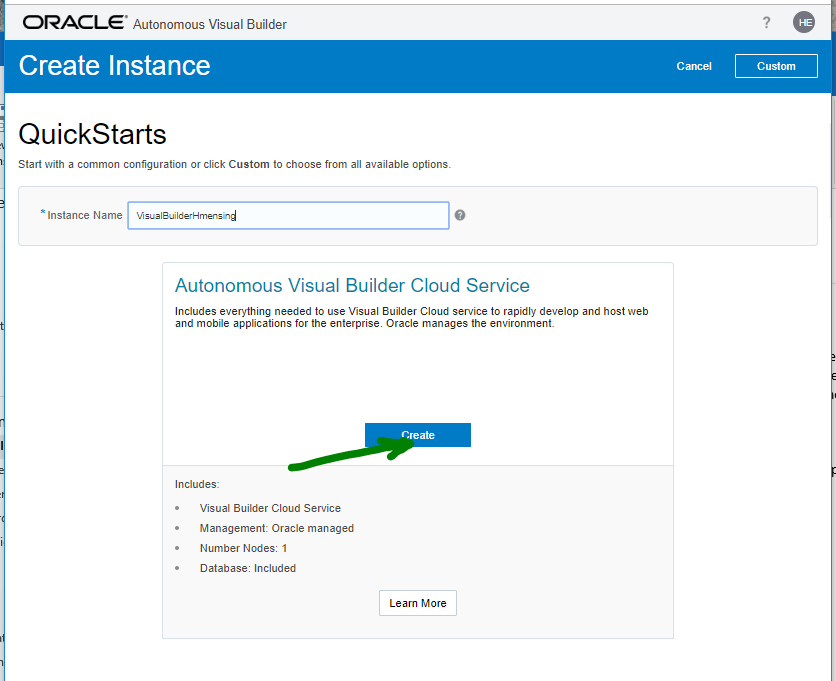


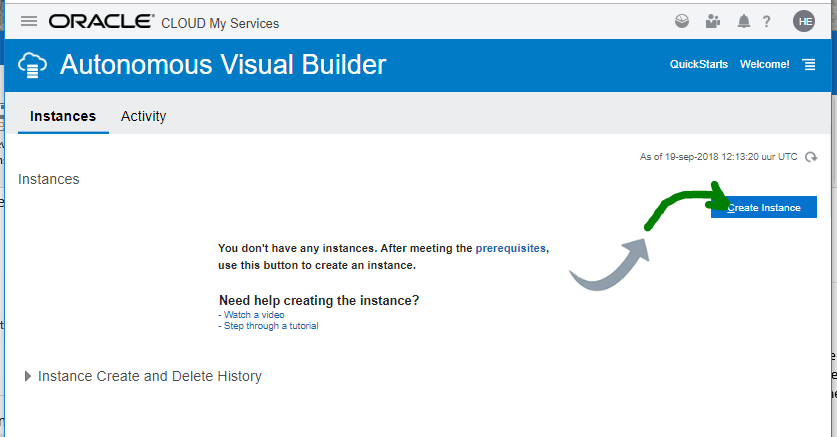


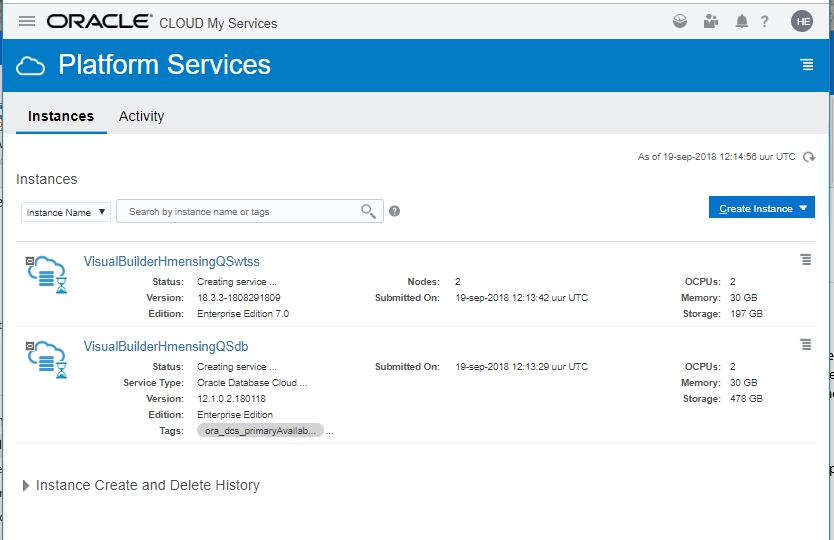




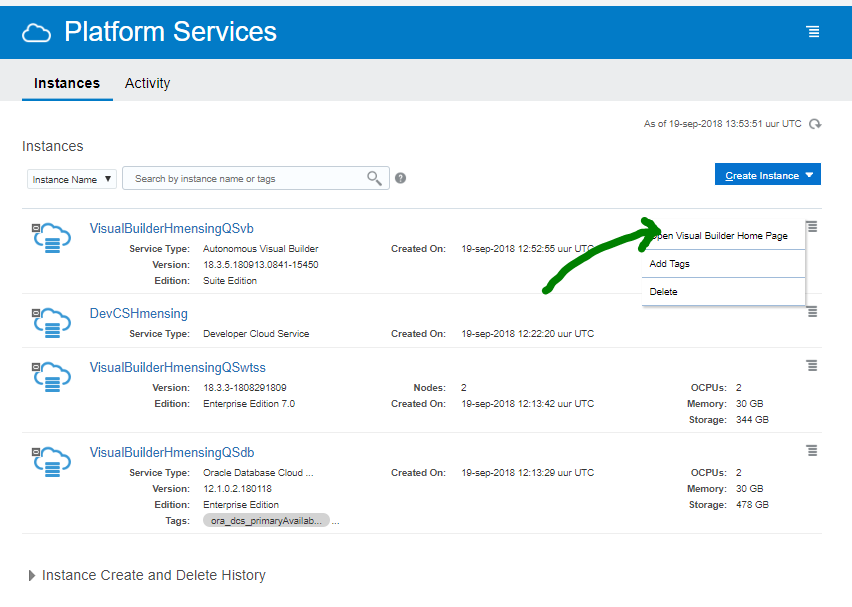


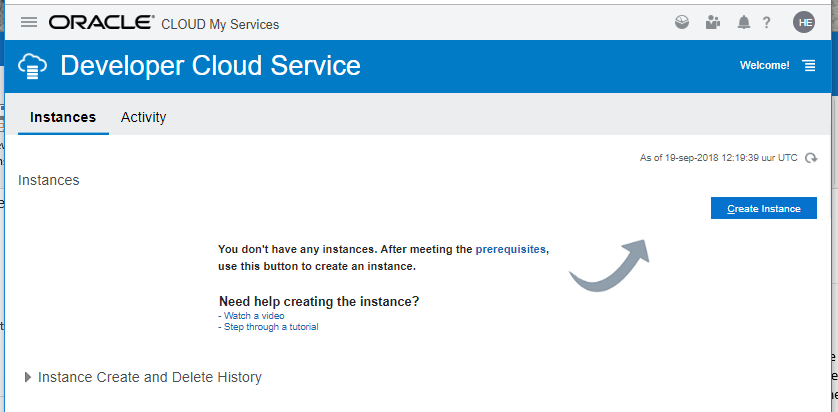


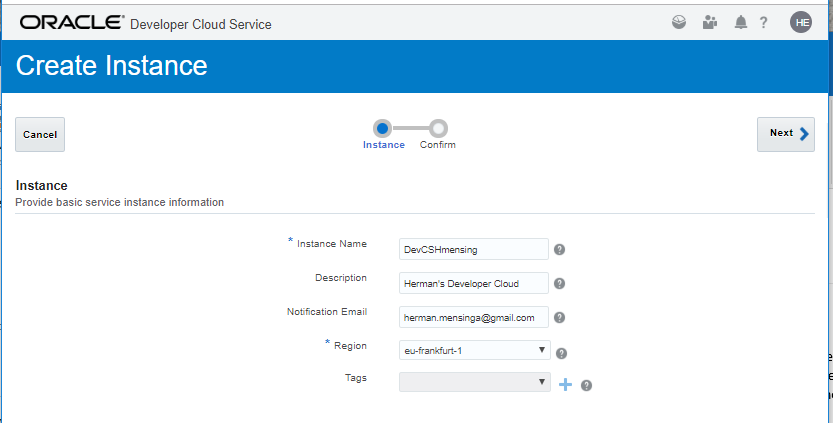


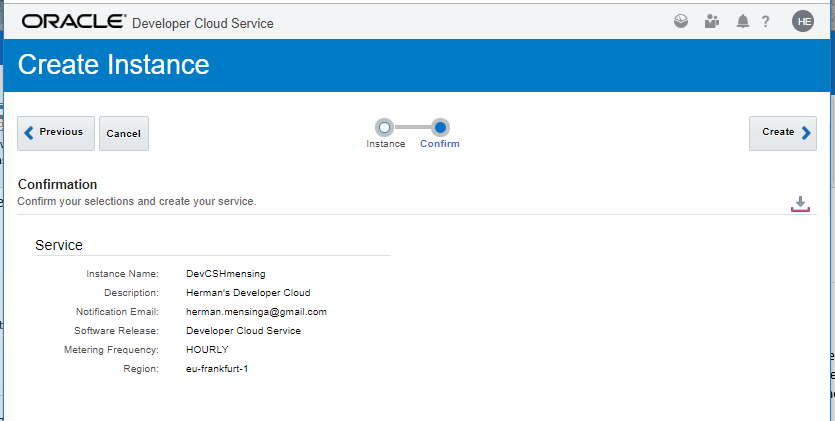


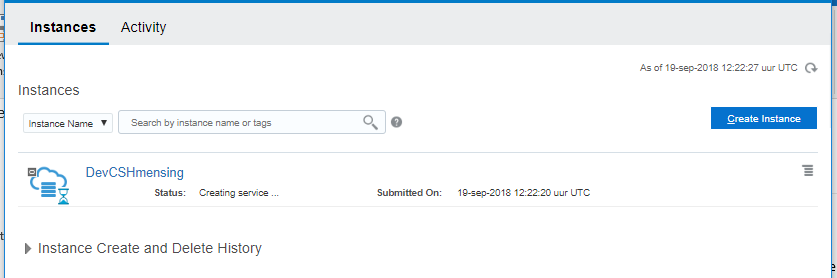
This takes 15 minutes



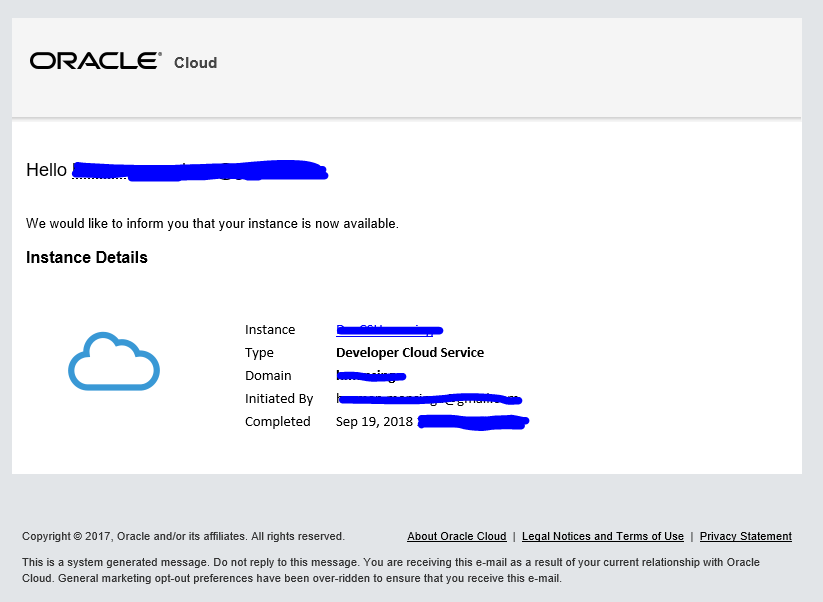


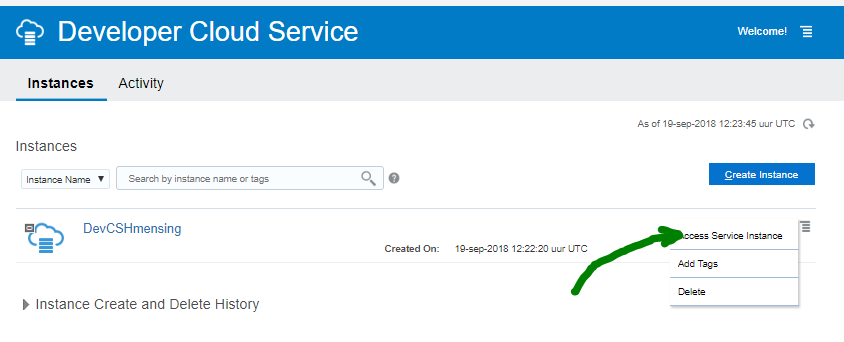


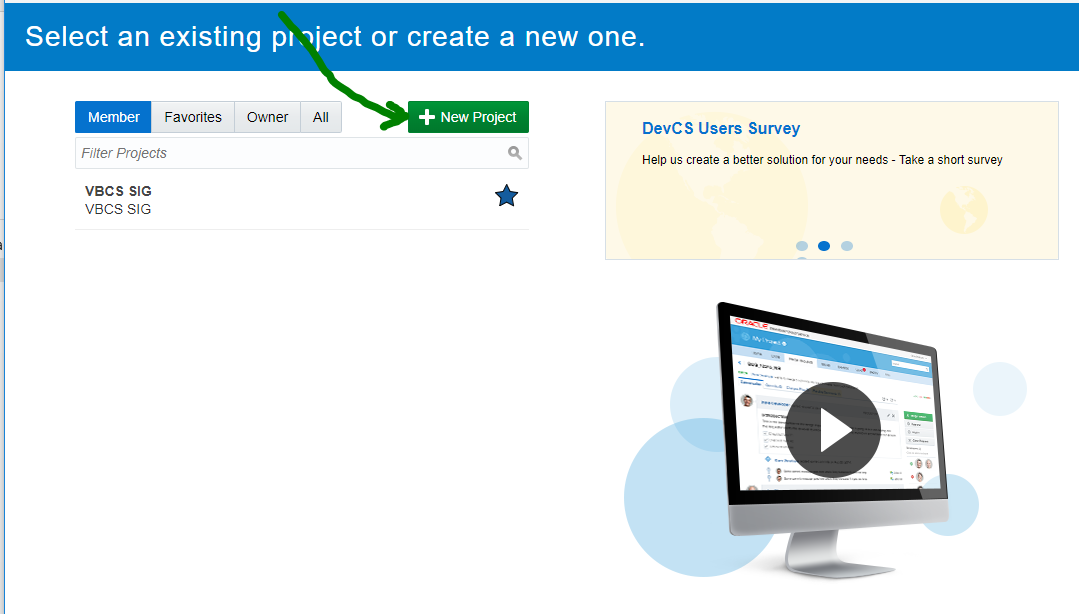


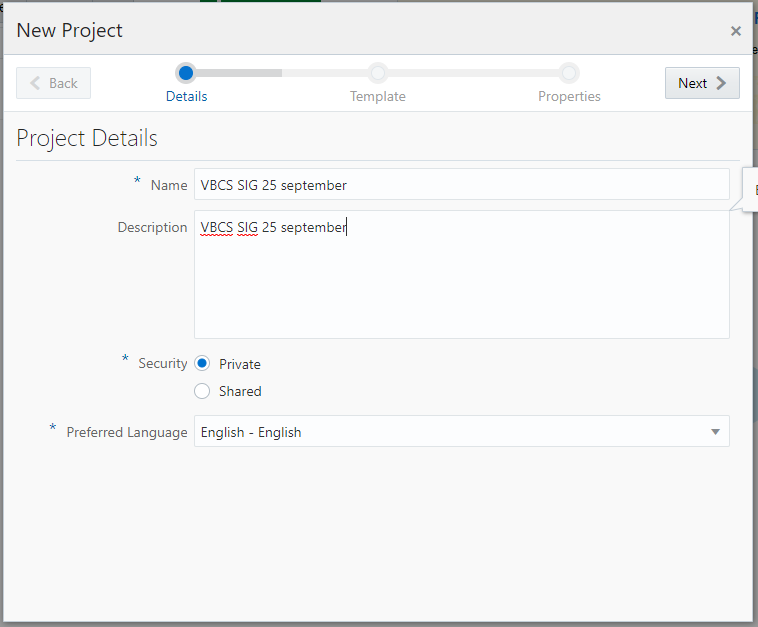


Mail

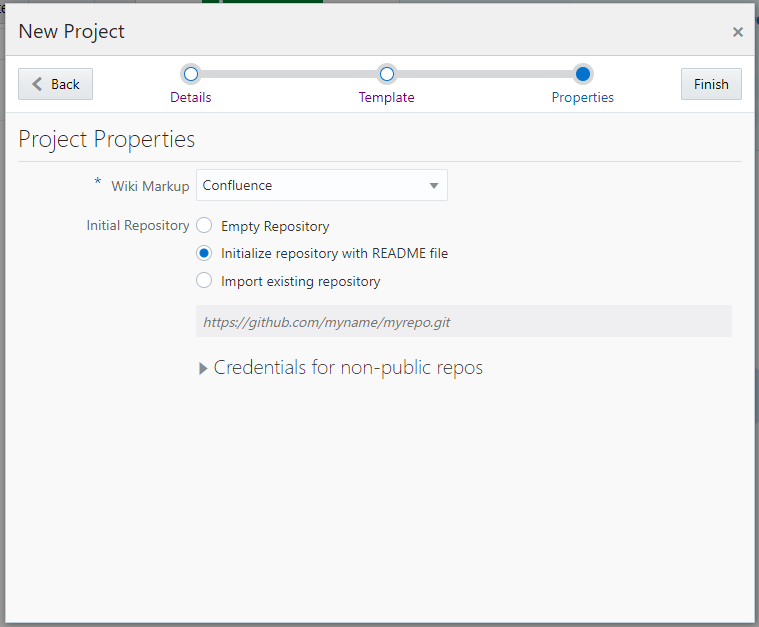


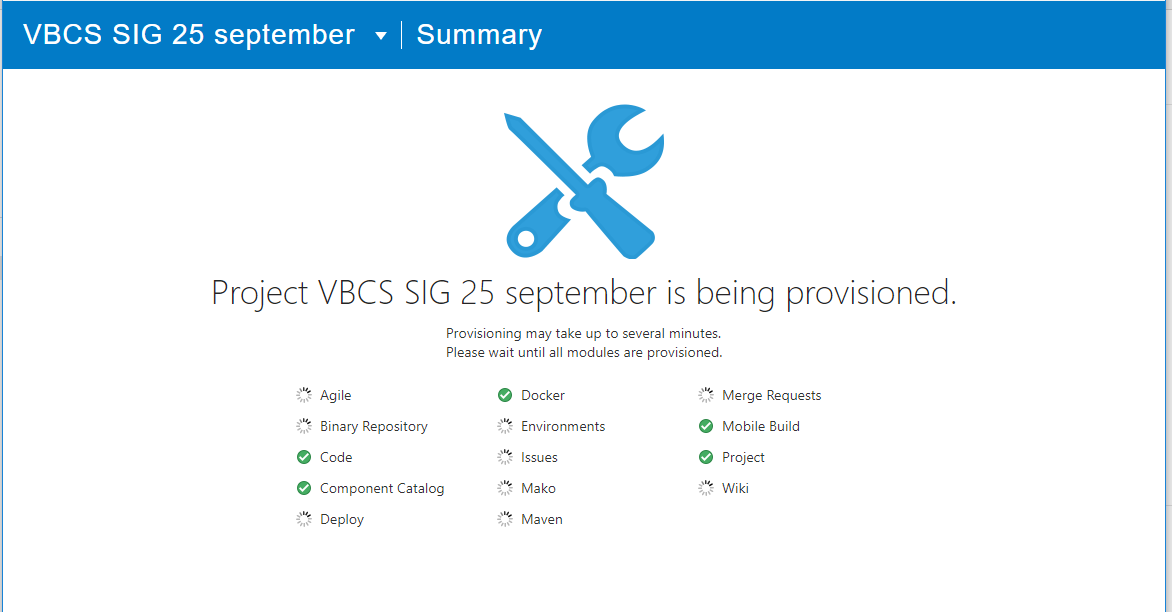


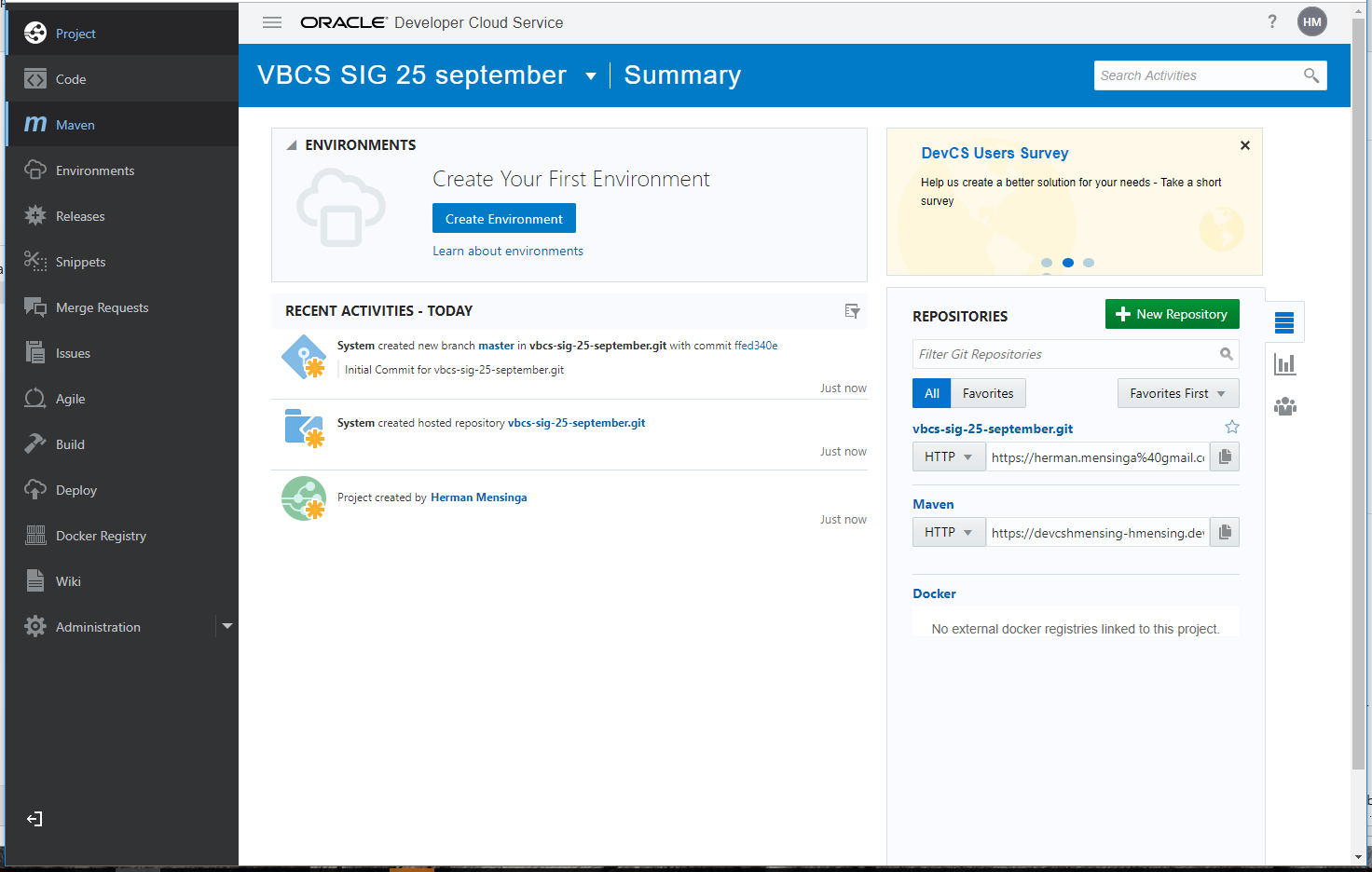












Clone it like you do normally 😊