# SAP HANA Product App

In this project you will create an app uses data on a UCC SAP HANA system

### Prerequisites

This case was prepared using version 7.21.0 of the desktop Mendix Modeler. You must also have an account in an SAP HANA system on which the database used in this case has been created.

## Create the App

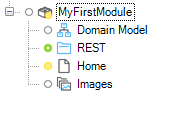
Create an app using the Fiori Blank SAP template. Name the app according to the format provided by your instructor.

## Create the Products Domain Model

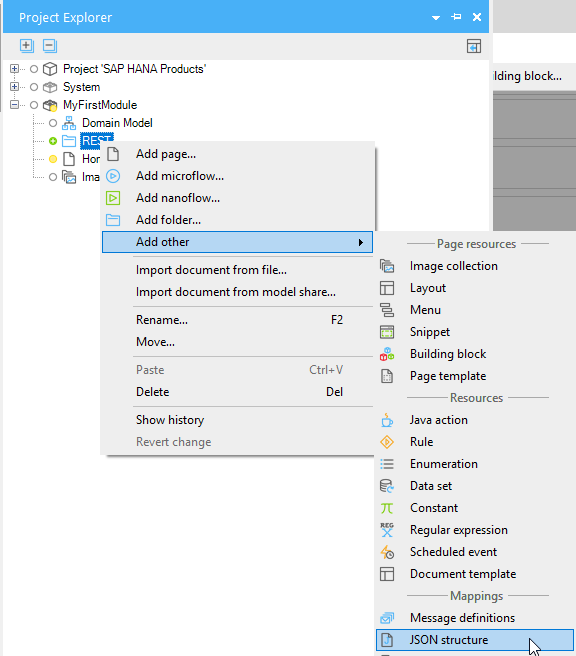
Even though the data in this case is accessed via SAP OData services, we cannot use the Mendix SAP OData Connector because the metadata returned from an SAP HANA system is not standard and Mendix Modeler cannot import the module created by the Mendix SAP OData Model Creator. As a result, we will use the standard Call REST activity. To use that activity in our microflows we have to create import mappings so that Mendix can parse the incoming data.

### Create the JSON Structure

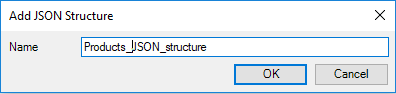
The Mendix Modeler can read XML data and create the necessary Domain model artifacts. Create a folder in your project by right-clicking MyFirstModule and selecting Add folder… Name the folder REST.



Right-click the new folder and create an JSON Structure.



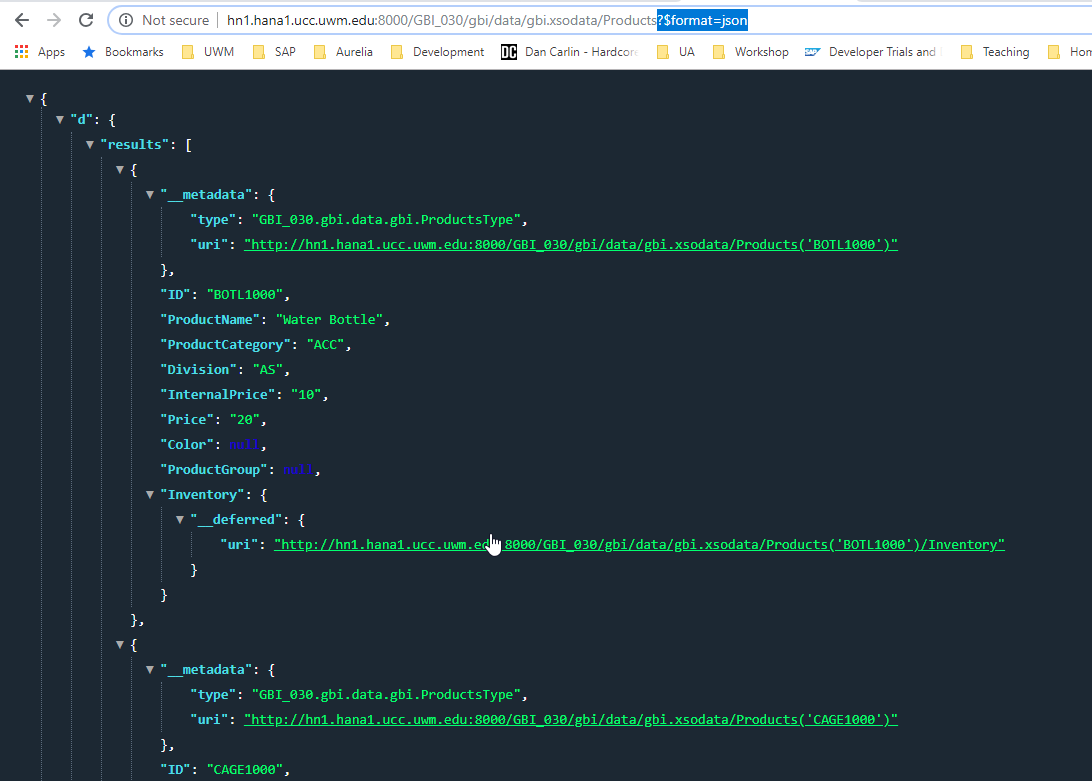
Name it Products\_JSON\_Structure.



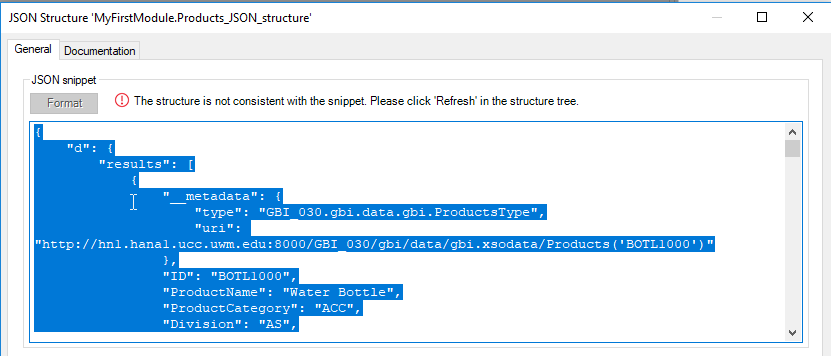
We can get the XML by copying the output from the service in a browser tab. Paste the service below into a browser tab (you should have logged into the SAP HANA system first).

http://<HOST>:8000/GBI\_030/gbi/data/gbi.xsodata/Products?$format=json

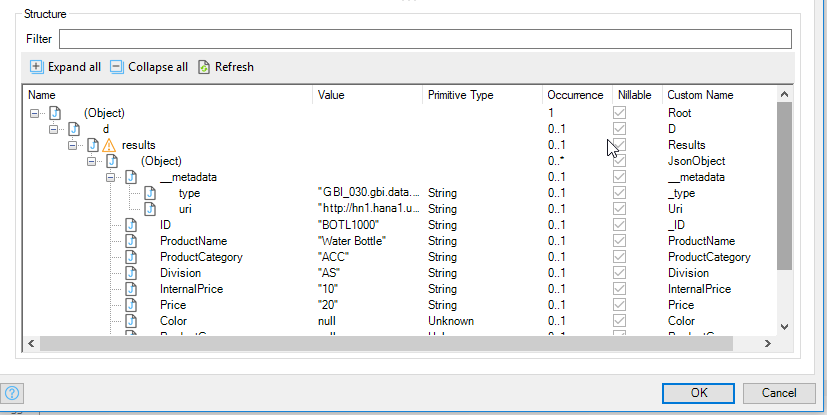
Replace <HOST> with your HANA system hostname.



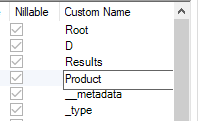
Copy the JSON (you can select all the XML by using ctrl-A) and paste it into the top field in the JSON Structure. Click Format to format the JSON.



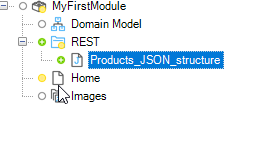
Click Refresh to update the Structure on the bottom of the JSON Structure dialog.



The names used in the Domain model artifacts that Mendix Modeler will create are in the right-most column. Locate the (Object) named JsonObject and double-click JsonObject to rename it Product.

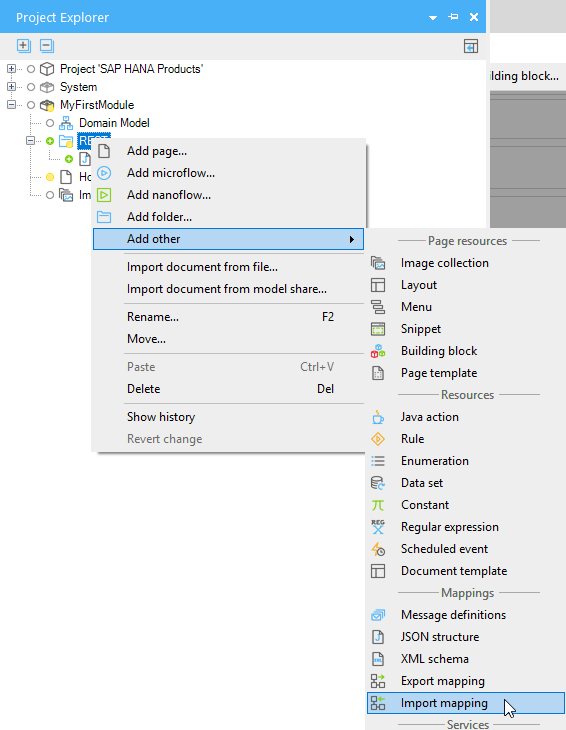


Click OK to save the JSON Structure.

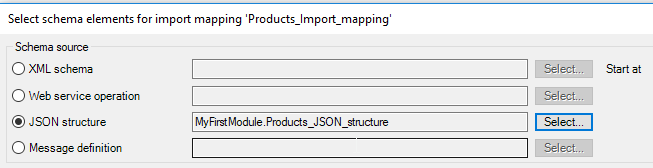


### Create the Import Mapping

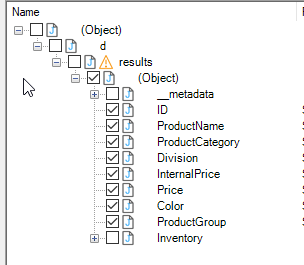
Now, right-click the REST folder and select Import Mapping.



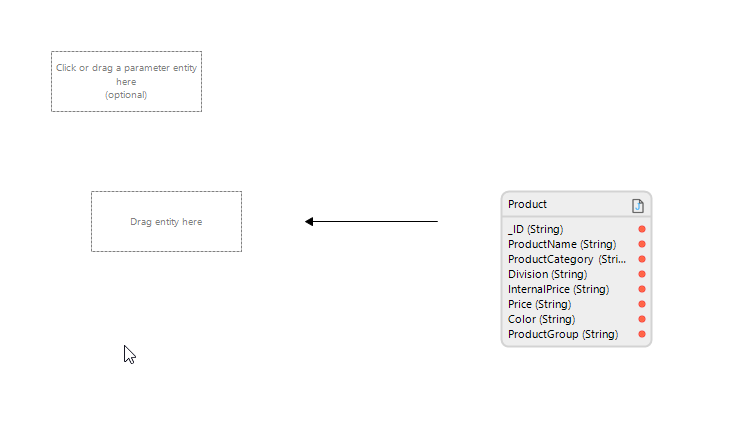
Name the Import Mapping Products\_Import\_Mapping. Select JSON structure and use the Select button to choose the Product\_JSON\_Structure.



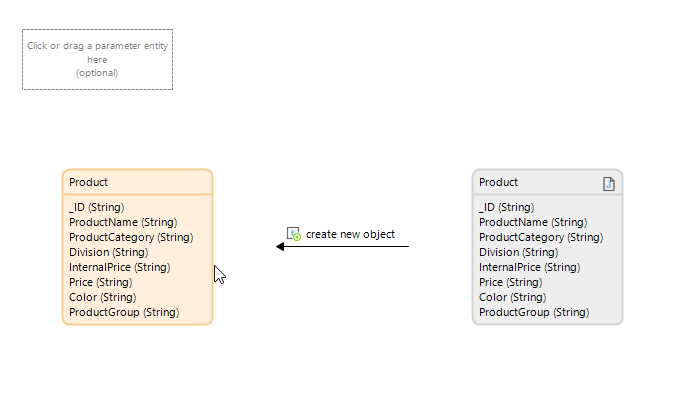
You select the properties you want to include in your Domain model in the bottom pane. Configure it as shown.



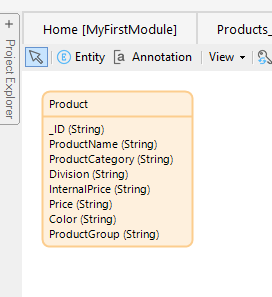
When you click OK it will look like this.



To create the entity in the Domain model, click Map automatically.

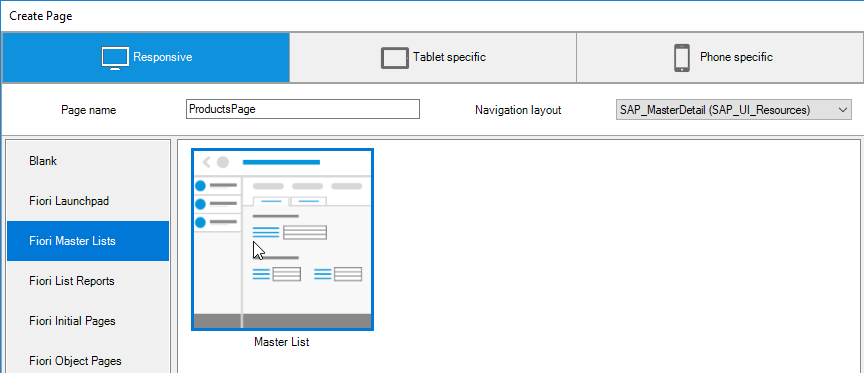


If you open the Domain model in MyFirstModule, you will see the entity.



## Create the Page

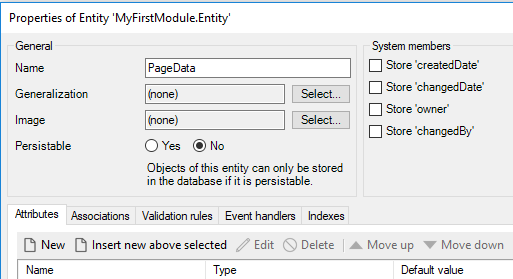
We’ll replace the Home page. Right-click MyFirstModule in the Project Explorer and select Add page… Configure it as shown below.



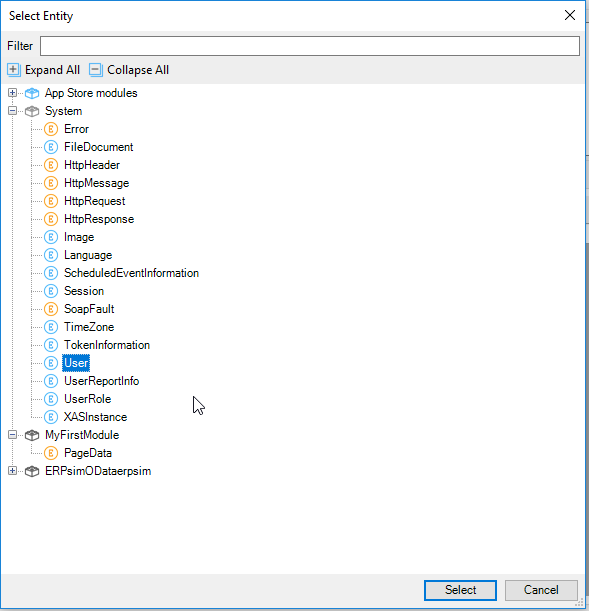
## Create the Microflow

### Configure the Domain Model

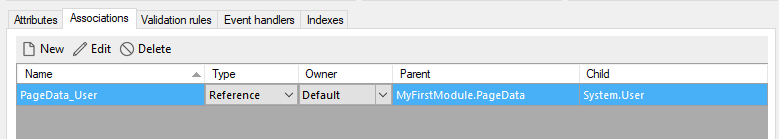
Open the Domain Model in the MyFirstModule module. Create a new Entity. Name it PageData and check No on Persistable. This prevents the data from being saved permanently in the database. We’ll use this entity to access the Product data through associations.



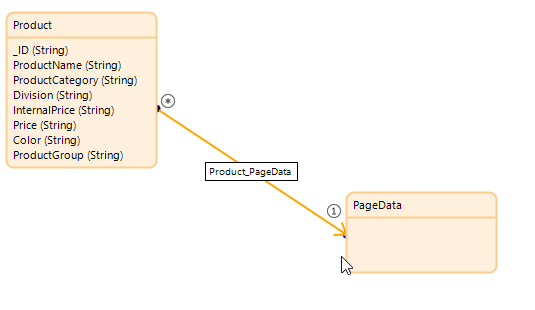
Add an association to the System User entity.



This association is necessary because PageData is a nonpersistable entity and we can’t retrieve directly from the database. We’ll use this association to retrieve it.



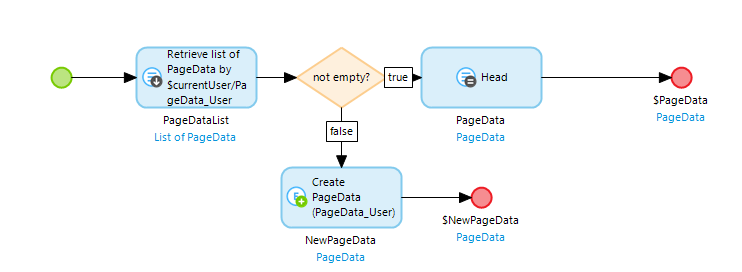
Once you save the PageData entity, hover over the border of the Product entity…



### Create the SUB\_GetPageData Microflow

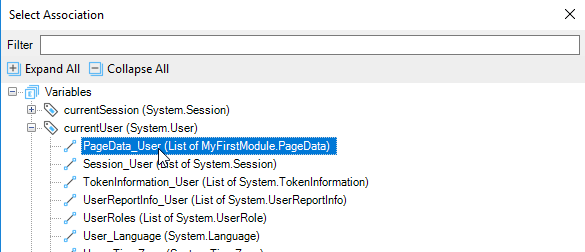
The PageData entity is used to access the data in Product data through the association we created so we have to retrieve it in order to create the association. We’ll create a submicroflow to do this. Right-click MyFirstModule in the Project Explorer and create a folder called Microflows. Right-click the Microflows folder and create a microflow called SUB\_GetPageData.

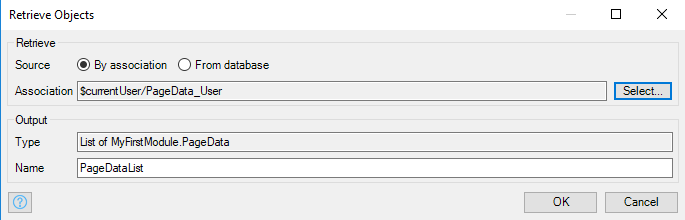
The microflow will look like this:



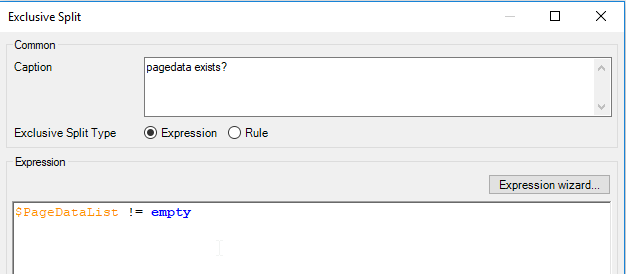
The first activity retrieves data from the PageData entity. If data already exists in PageData then we take the first record. If there isn’t any data yet, we create a record.

Start by adding an activity and making it a Retrieve type. Select association for the Source and navigate to the PageData\_User association that we created earlier.

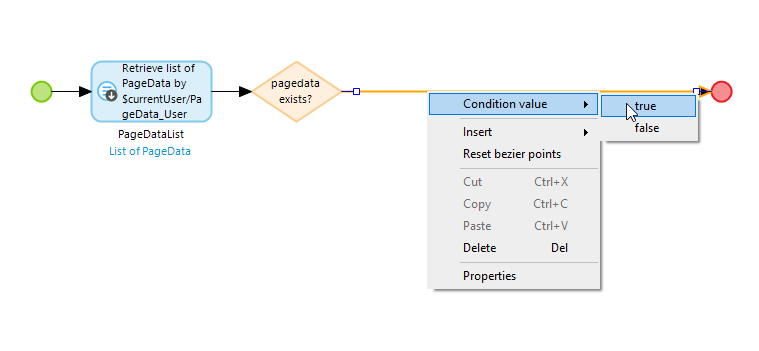




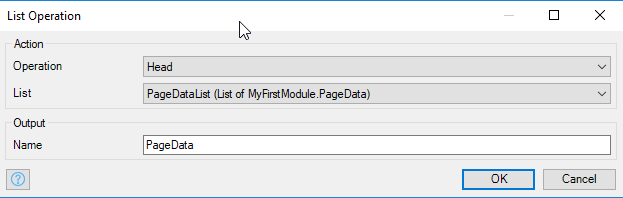
Add an Exclusive Split and configure it like the image below. This checks to see if the PageDataList is empty.



Right-click the line to the right of the exclusive split and select the true condition.

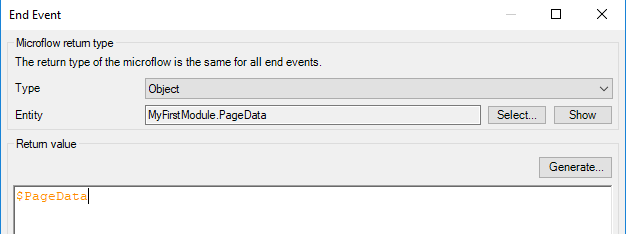


Insert an activity to the right of the exclusive split and configure it as a List Operation type like the image below:

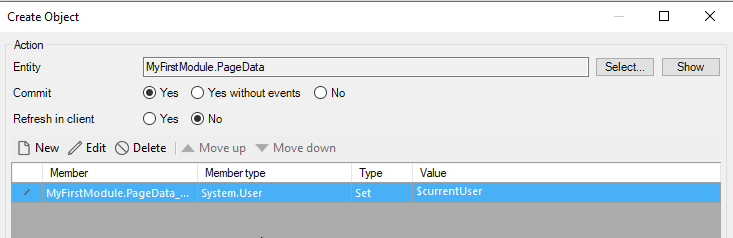


This will select the first item on the list (there can only be one item) and creates an object called PageData.

Double-click the End event and configure it as shown. This will pass the PageData object to parent microflow.

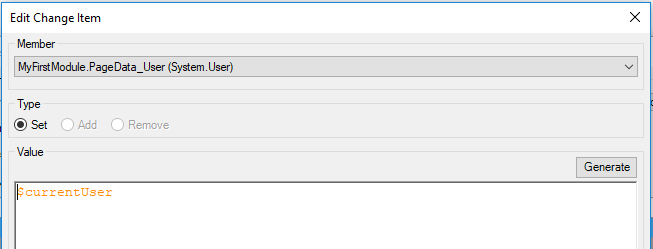


To configure the false condition on the exclusive split, drag down from the bottom point of the exclusive split diamond and select Activity when you let the mouse button up. Configure the activity as a Create object activity as shown:

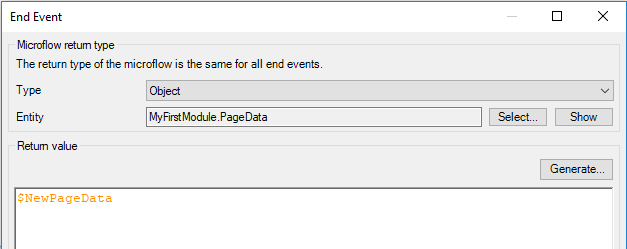


The Member creates the association between the new PageData object and the system user entity. Make sure you select Yes for Commit.

Click New to create a new member and configure it as shown below. The currentUser variable is a built-in variable to contains the current user of the app.

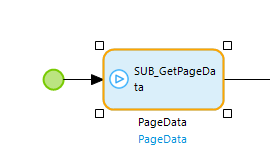


To finish the microflow, drag out from one of the sides of the Create object activity and create an End event. Configure it as follows:



### Create the GetProducts Microflow

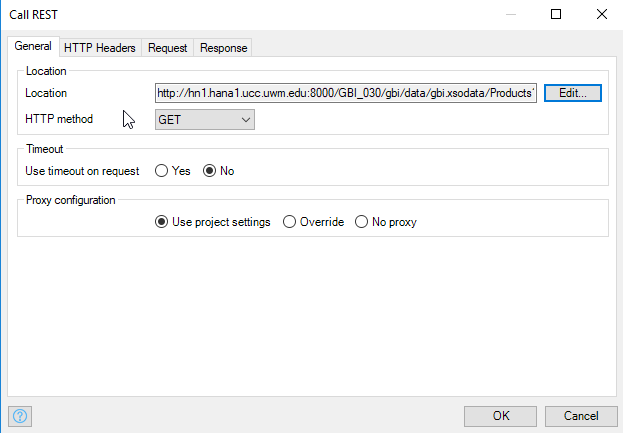
Create a new microflow in the Microflows folder called ACT\_GetProducts. Drag the SUB\_GetPageData onto the new microflow just to the right of the Start event. Double-click it to open it and then click OK.



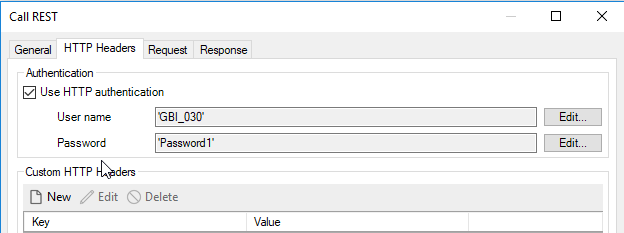
Add another activity of type Call REST. Configure the General tab as shown. The URI to use in the location is:

http://<HOST>:8000/GBI\_030/gbi/data/gbi.xsodata/Products?$format=json

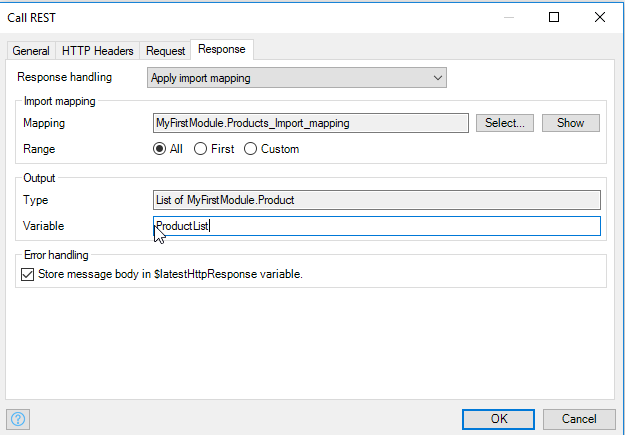
Replace <HOST> with your SAP HANA’s hostname.



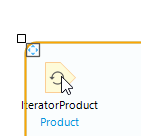
Configure the HTTP Headers tab as shown below.



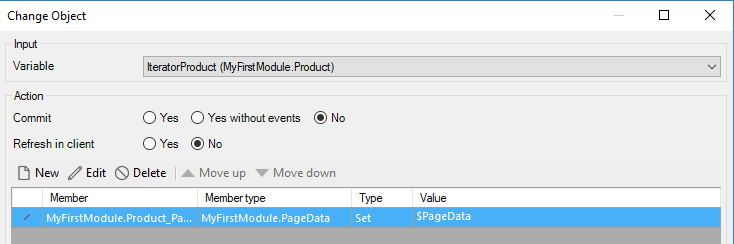
Configure the Response tab as shown below.



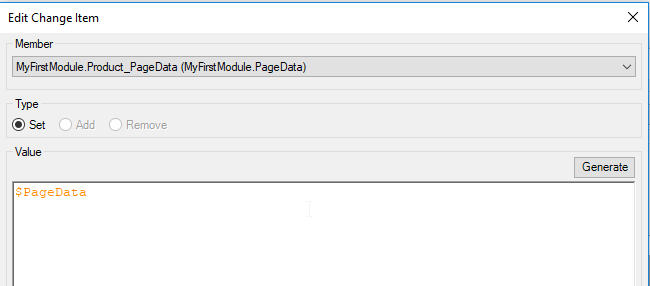
Next, we have to create the associations between the records in the ProductList and PageData. Add a loop to the microflow and configure the iterator to use the ProductList.



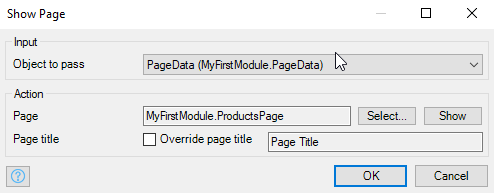
Add a Change object activity inside the loop and configure it as shown below.



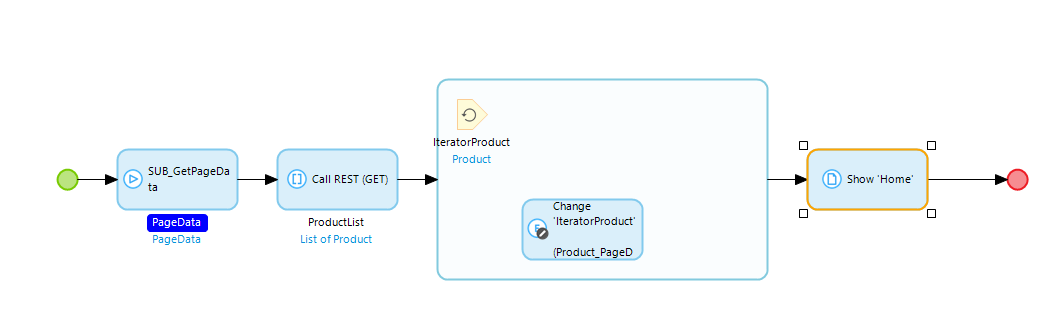
The details for the configuration of the Member is shown below.



Finally, add a Show page activity between the loop and the End event. Configure it to open the ProductsPage page and pass the PageData object.

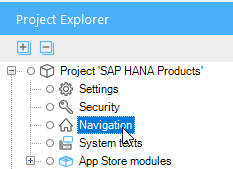


The final microflow looks like this:

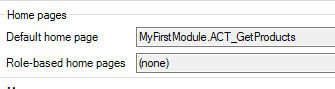


### Add the Page to the Navigation Model

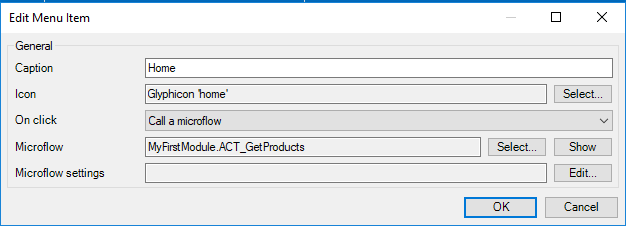
Open Navigation.



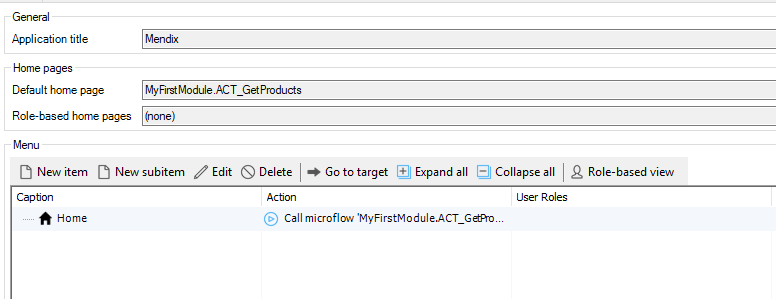
Configure the Default home page to be the ACT\_GetProducts microflow.



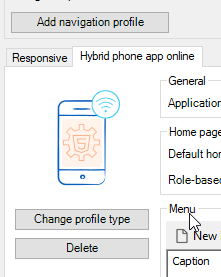
Also configure the Home menu item to open the microflow.



The Navigation should look like this:



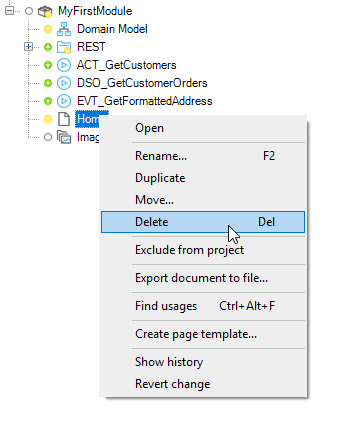
Select the Hybrid phone app online tab and delete the navigation profile.



## Configure the Page

### Delete the Home Page

Delete the Home page in the Project Explorer.



### Configure the ProductPage

Open the ProductPage. The page includes a Listview on the left side (the master side) and a Data view on the right side (the detail side). Double-click the header of the Listview to open its properties.

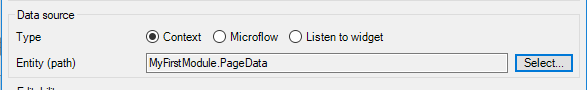
Our microflow passes the PageData object which we can’t bind to the Listview so add a Data view to the master side by clicking the Data view at the top of the screen and clicking just above the Listview.

|  |  |
| --- | --- |
|  |  |

Now drag the Listview inside the Data view.

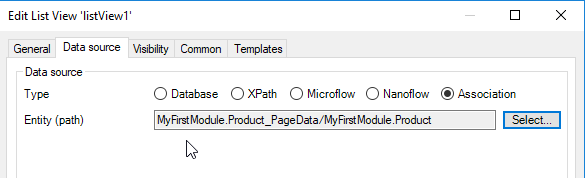


Double-click the header of the Data view and configure the Data source as shown below.



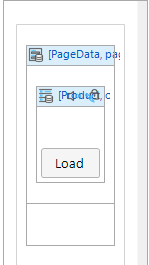
When prompted to automatically fill the contents, choose no.

Now double-click the Listview header and configure the Data source as shown below.

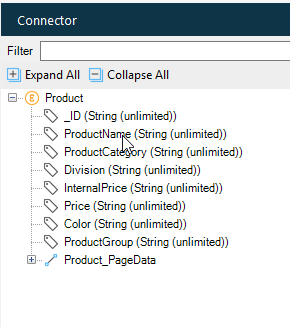


Choose not to automatically fill the contents.

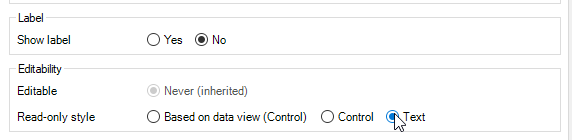
Delete the widgets inside the Listview.



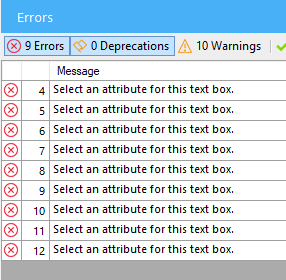
If you click inside the Listview you can see the available fields in the Connector view.



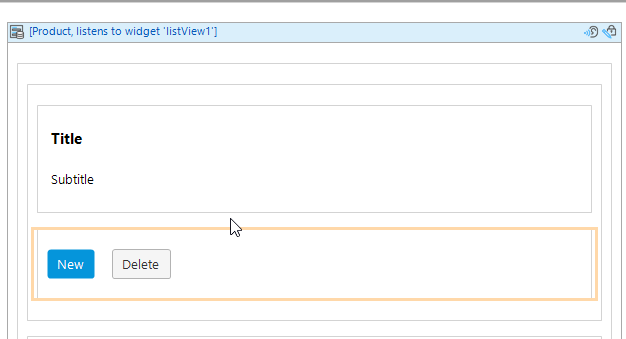
Drag ProductName into the Listview then double-click the widget and configure it as shown below.



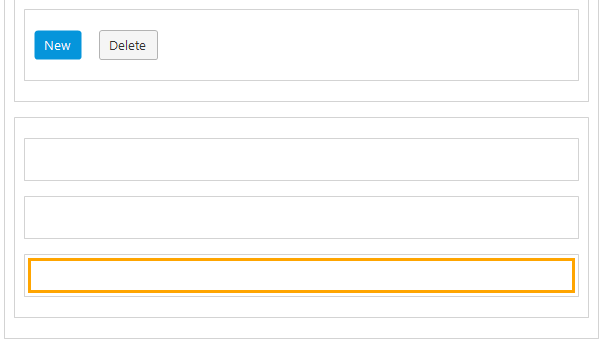
If you look at the Error tab you will see a lot of errors.



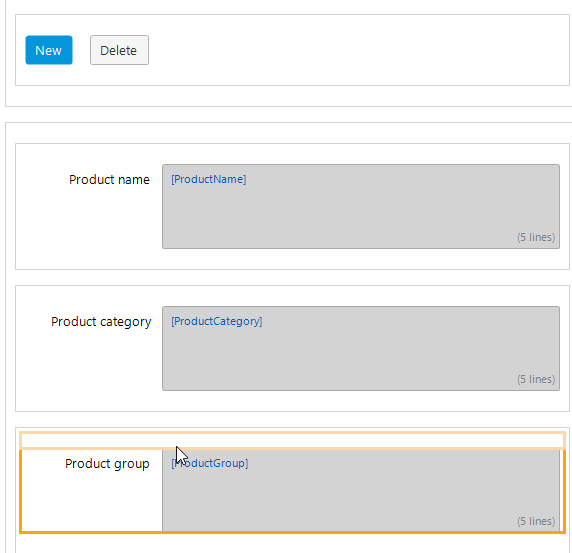
All of the errors are because there are widgets on the detail side of the ProductPage that have not been configured with a data source. The detail side is embedded in a Data view that is configured to listen to the Listview. This means that it will be bound to whichever Product object that is selected in the Listview.



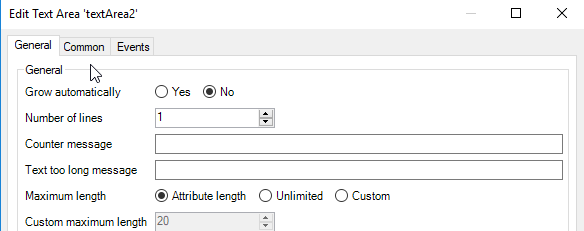
Delete the Text widgets that are currently configured to show Category and Value.



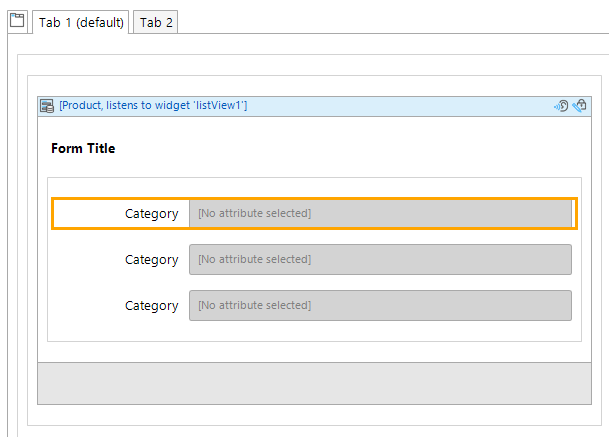
If you click inside one of these containers you can see the available fields in the Connector view. Drag the following fields onto the page.



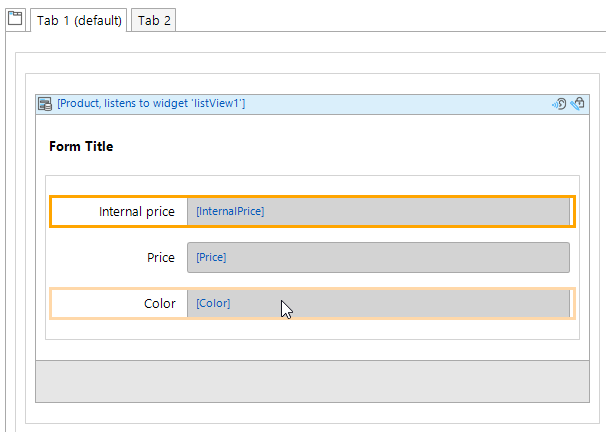
Mendix didn’t know the size of the fields so it add Text Areas with 5 lines. Configure each of these to change the lines to 1.



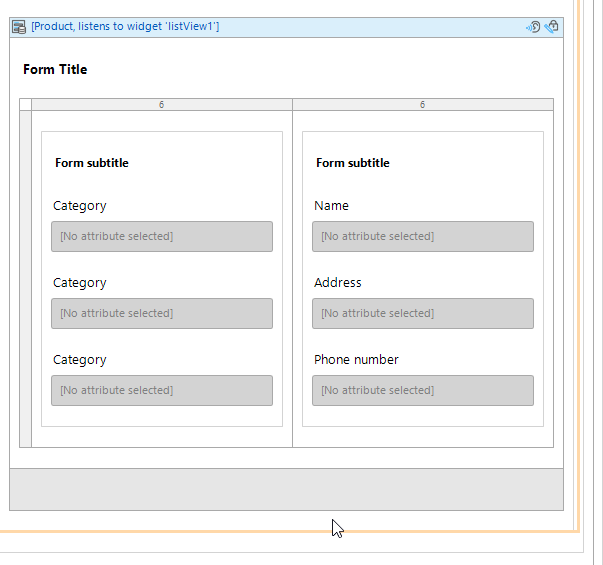
Next, locate the tab control.



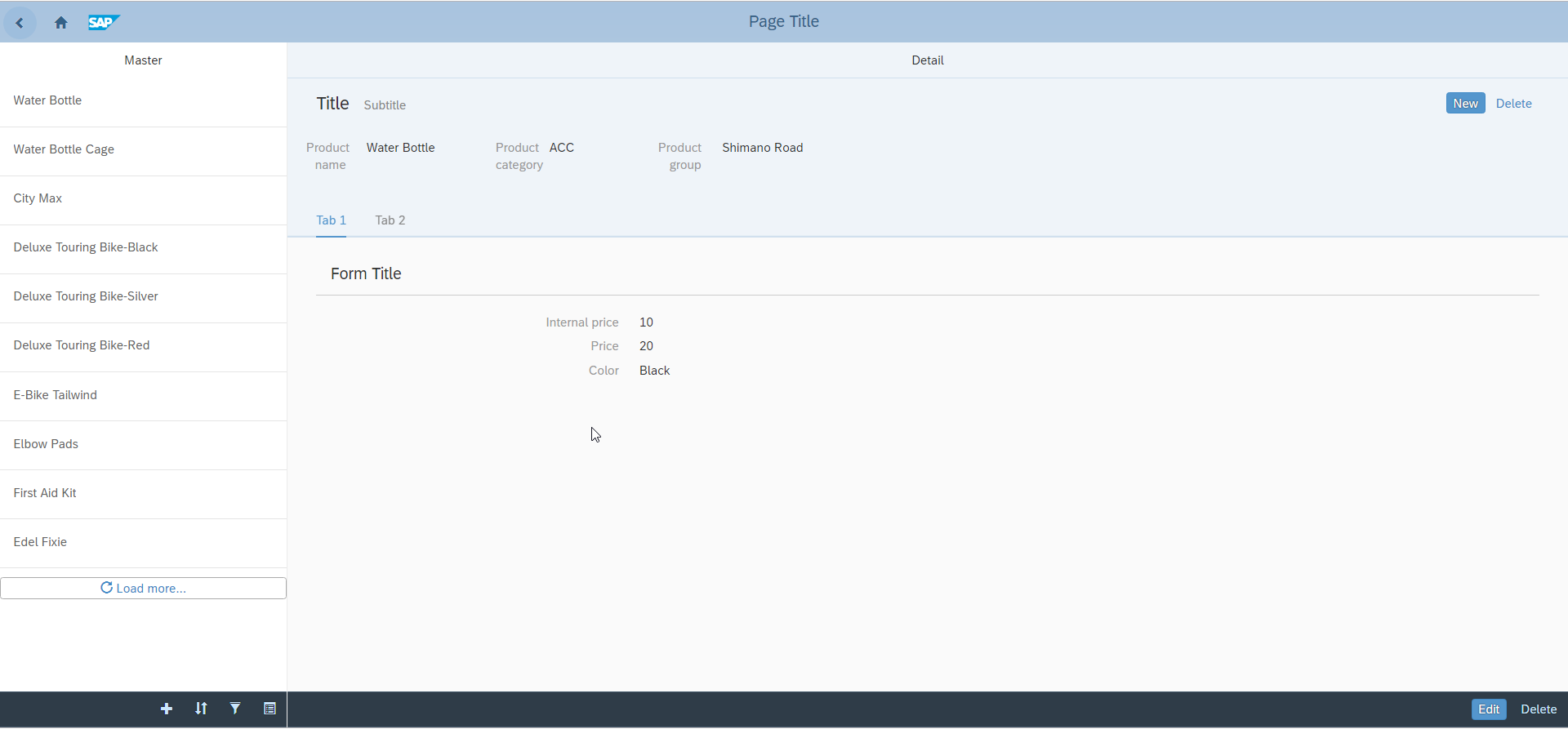
Drag and drop the following fields onto the three existing fields.



Finally, delete the Data view at the bottom of the detail side.



There should be no errors remaining. If you run the app it should look like this:

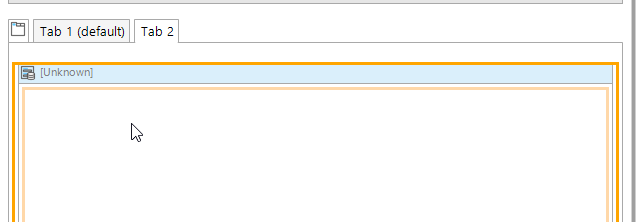


## Add the Inventory

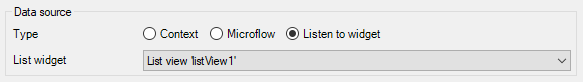
Next, well retrieve inventory data on the products.

### Configure the ProductsPage

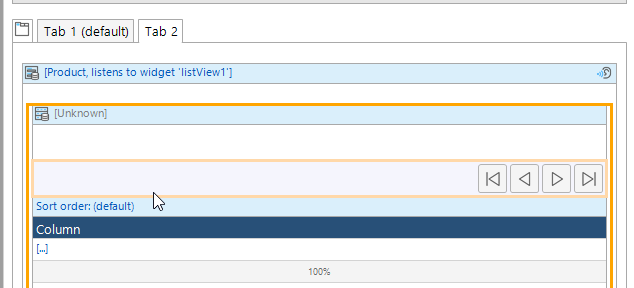
Locate the tab control on the detail side of the ProductsPage and select Tab 2. Add a Data view to it by clicking the Data view at the top of the Modeler and clicking on the page.



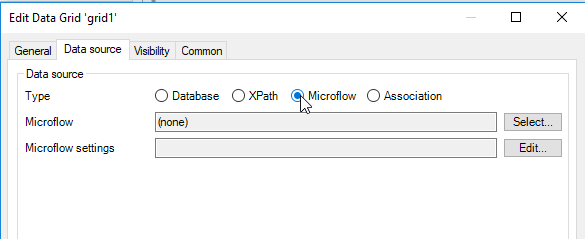
Configure it to listen to the Listview.



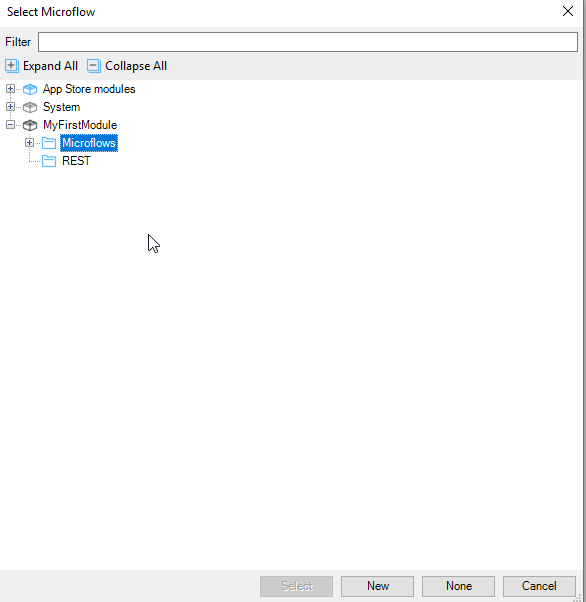
Add a Data grid widget inside the Data view.



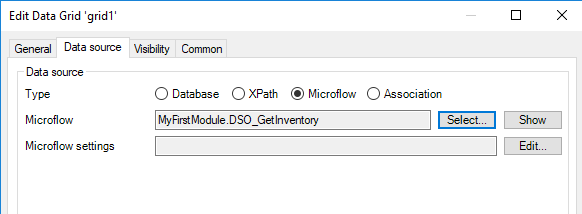
Configure the Data source as shown below.



Click Select… for the Microflow. Select the Microflows folder and click New.



Name the microflow DSO\_GetInventory. Click Show then OK.



### Create the Import Mapping

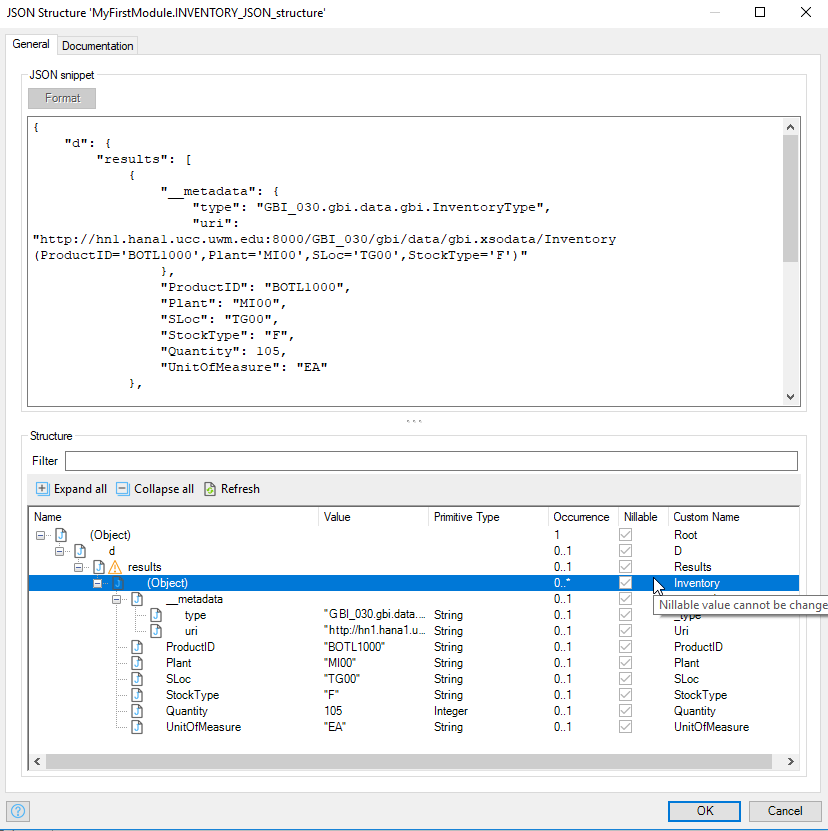
To retrieve inventory data we have to create an Import mapping for the data. The URI used to retrieve inventory is:

http://<HOST>:8000/GBI\_030/gbi/data/gbi.xsodata/Products('BOTL1000')/Inventory/?$format=json

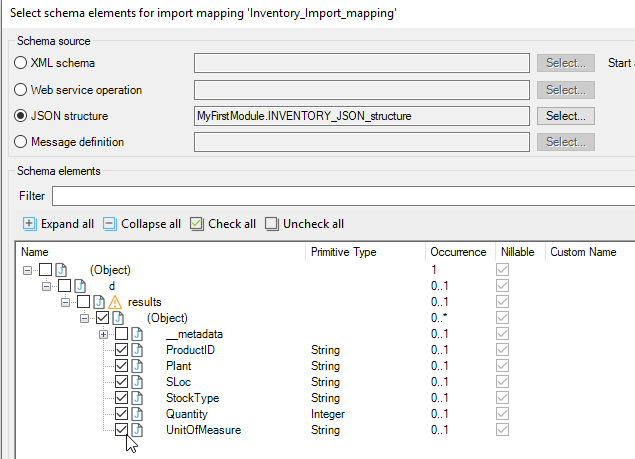
Where <HOST> would be your SAP HANA hostname and BOTL1000 would be the product for which you want the inventory data.



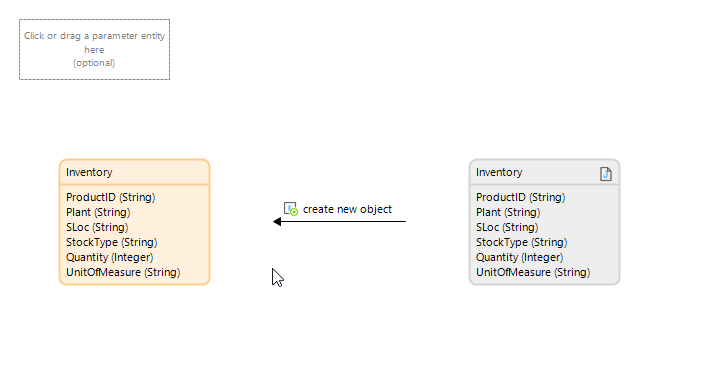
Create a JSON structure in the REST folder called Inventory\_JSON\_Structure and copy the JSON into it. After clicking Refresh, locate the (Object) and rename it Inventory.



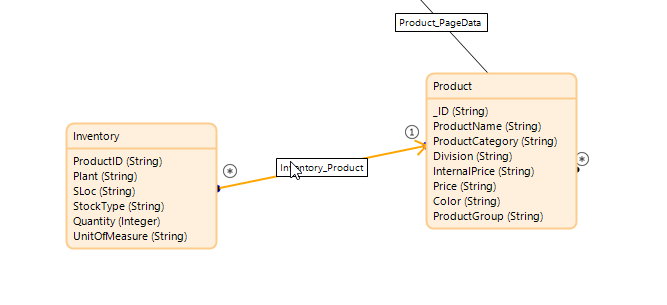
Create an Import mapping in the REST folder called Inventory\_Import\_Mapping and configure it as shown below.



After mapping the entity is created.



In the Domain model create an association between Inventory and Products by hovering on the border of Inventory then clicking and dragging to Producct.



### Configure the Microflow

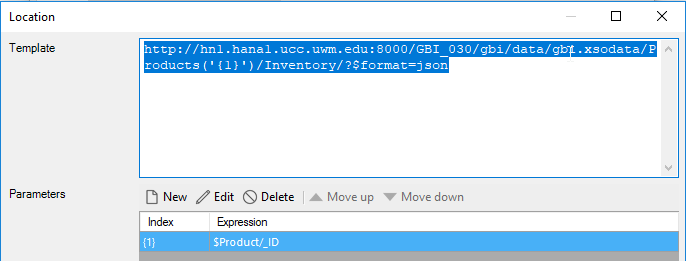
Open the DSO\_GetInventory microflow. The microflow was created with the Product parameter because the Data grid was embedded in the Data view that was bound to Product. This parameter will be equal to the Product selected in the Listview.



Add a Call REST activity. The URI to use is:

[http://hn1.hana1.ucc.uwm.edu:8000/GBI\_030/gbi/data/gbi.xsodata/Products('{1}')/Inventory/?$format=json](http://hn1.hana1.ucc.uwm.edu:8000/GBI_030/gbi/data/gbi.xsodata/Products('%7b1%7d')/Inventory/?$format=json)

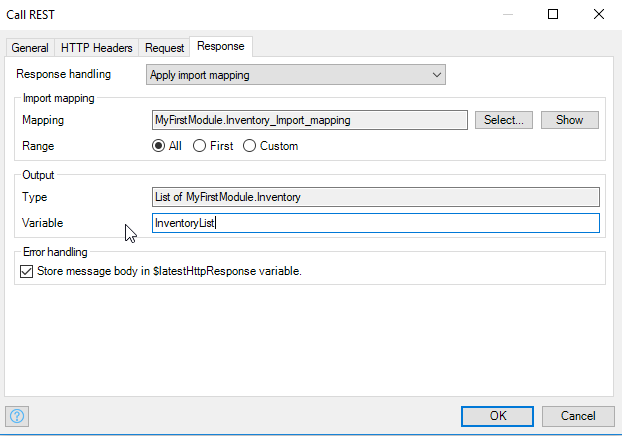
The {1} is a parameter where we can substitute the ProductName from the Product parameter.



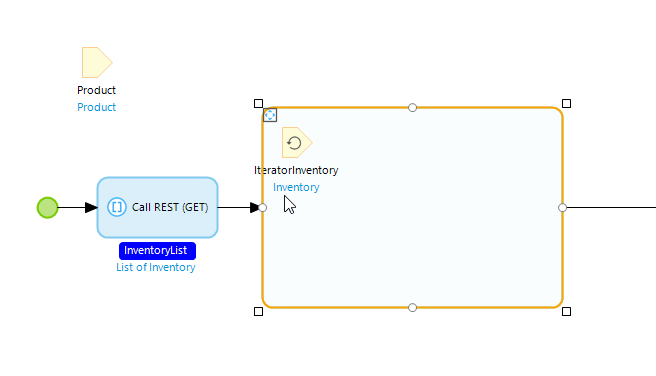
The details for the configuration of the parameter is shown in the image below.



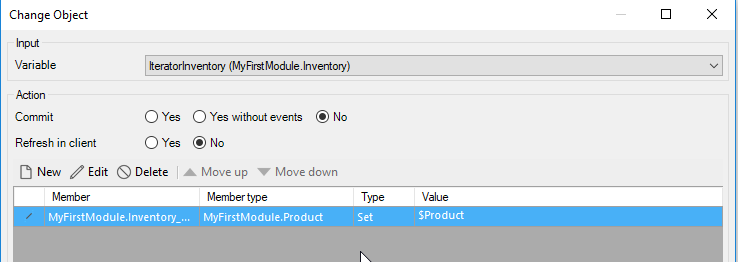
Configure the HTTP Headers tab as you did for the Products and configure the Response tab using the Inventory\_Import\_Mapping.



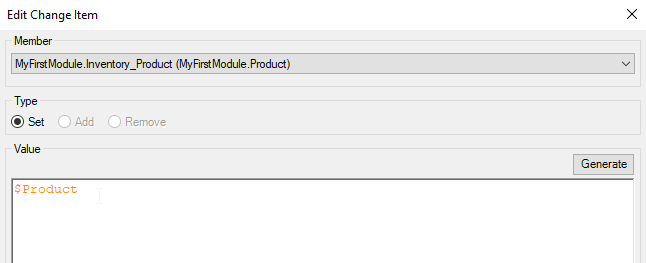
Add a loop and configure the iterator to iterate over the InventoryList.



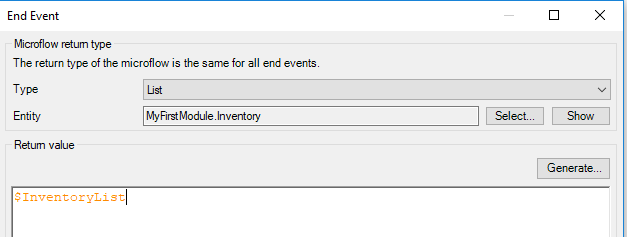
Add a Change object activity inside the loop and configure it to change the IteratorInventory object as shown below.



The details of the Member configuration is shown below.

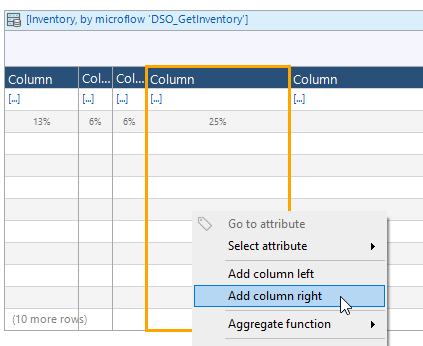


Double-click the End event and configure it as shown.



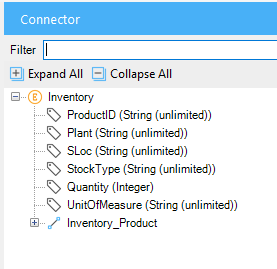
### Configure the Inventory Grid

Return to the Data grid and add five columns by clicking inside the existing column and clicking Add column right five times.

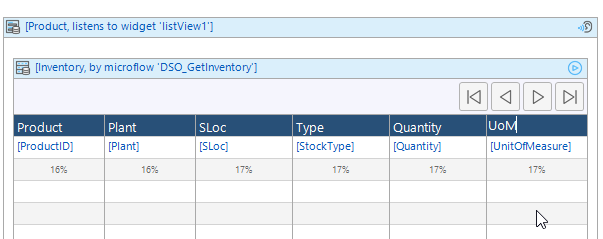


You can equalize the column widths by right-clicking on the Data grid header and selecting Reset column widths.

If you click inside a column, you will see the available fields in the Connector view.



Drag the fields into the columns as shown below. If you select a column and begin typing, it will set the column header.



Now when you run the app, you will see the inventory data on the second tab.

