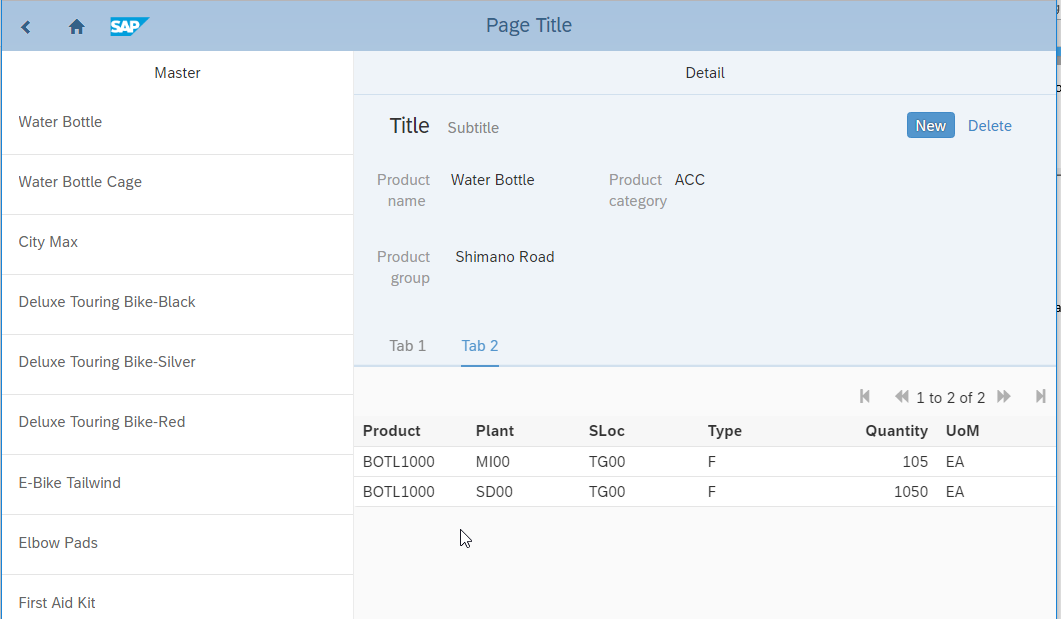
# Adding Create, Update and Delete to the Products App

## Prerequisites

This builds on the SAP HANA Product App so that case must be completed before starting this one.

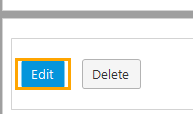
## The Products App

At the end of the SAP HANA Product App case the app retrieved Product and Inventory data.

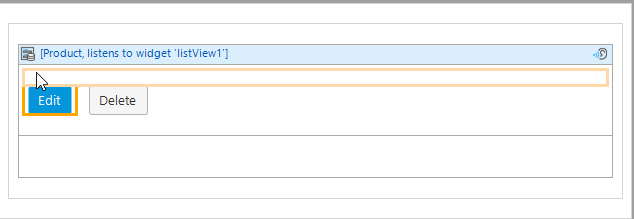


## Create the Edit Form

Locate the Edit and Delete buttons at the bottom of the ProductsPage.



Add a Data view to the footer (just above the buttons) that listens to the Listview widget then drag the buttons into it. This is necessary because when we click the edit button we need to pass the selected product to the edit page.



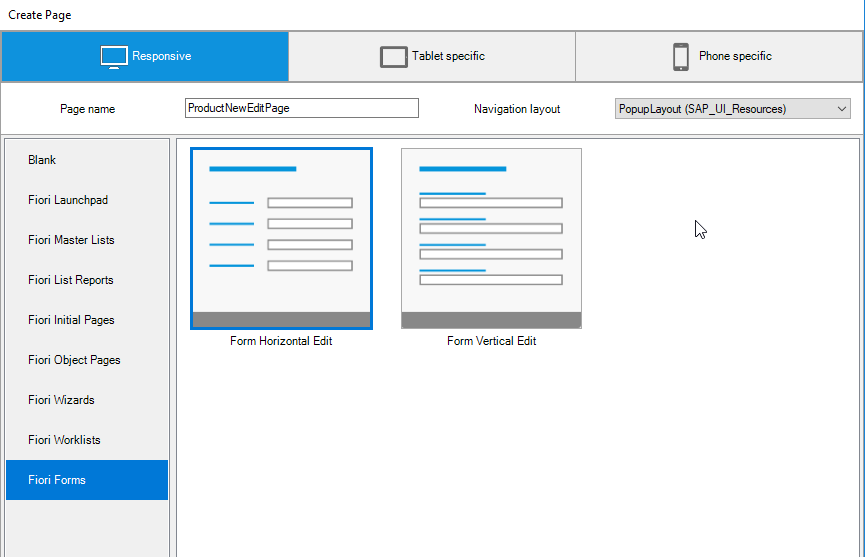
Right-click the Edit button and select Generate on click page… Configure the new page as shown below:

Page Name: ProductNewEditPage

Navigation layout: PopupLayout

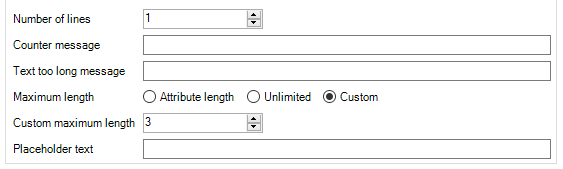
Template category: Fiori Forms

Template: Form Horizontal Edit

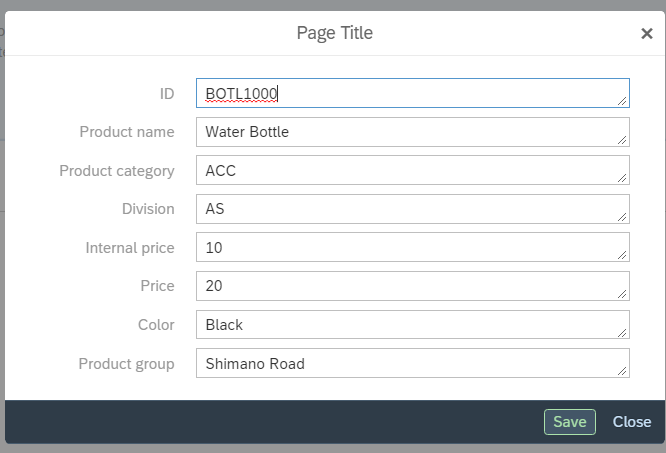


The page is created but we need to configure it a bit. For one thing, Mendix doesn’t know the actual size of the fields so we need to adjust the size. Configure each field to be 1 line and then configure the widths of the fields as shown in the table below. The image below the table gives an example.

|  |  |
| --- | --- |
| ID | 10 |
| ProductName | 40 |
| ProductCategory | 3 |
| Division | 2 |
| Color | 10 |
| ProductGroup | 20 |



Now when you run the app and click the Edit button, the product opens in the popup.

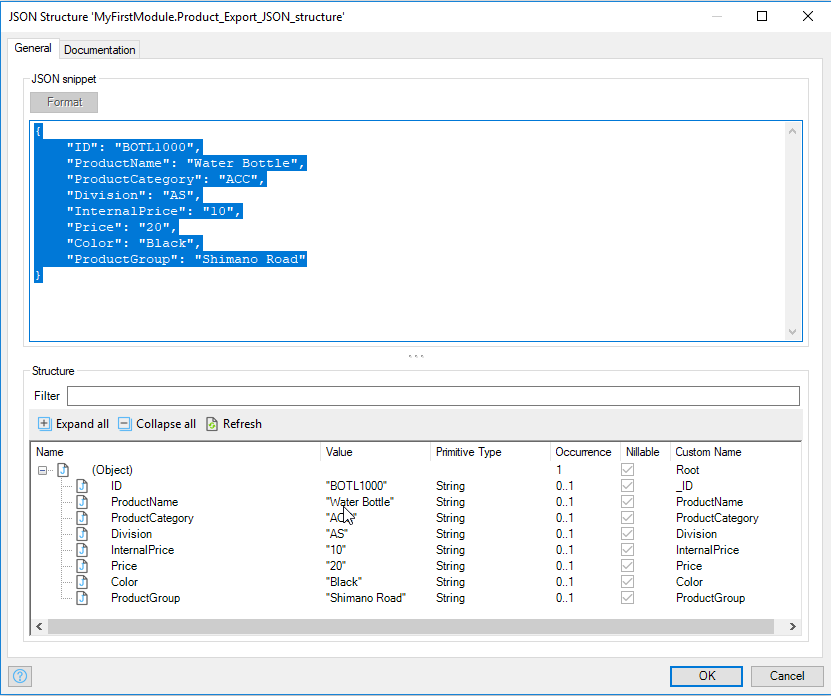


## Create the Export Mapping

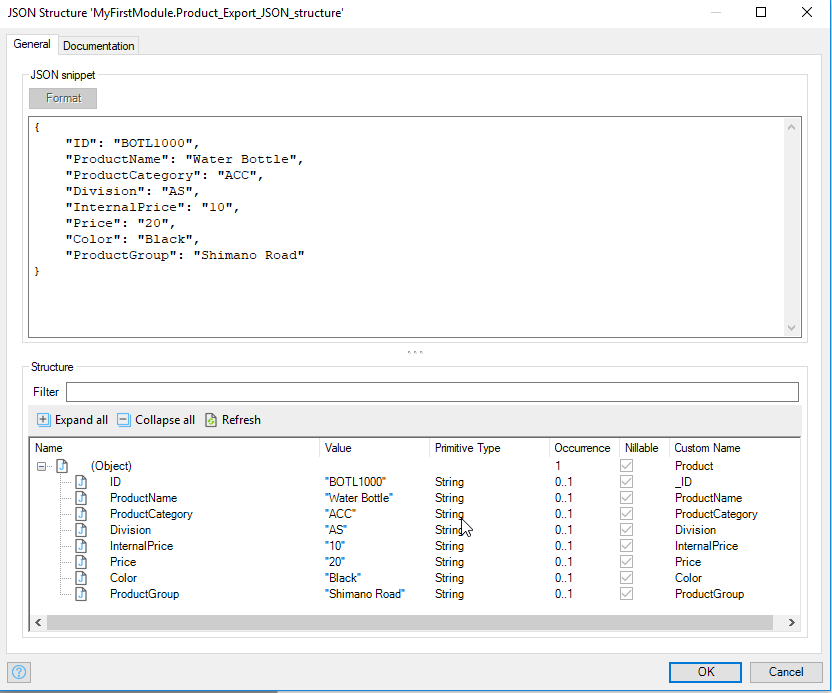
Just as we created an Import Mapping to retrieve data from the server, we must create an Export Mapping to send data to the server. Create a JSON Structure in the REST folder. Paste the JSON below as the JSON snippet.

|  |
| --- |
| {  "ID": "BOTL1000",  "ProductName": "Water Bottle",  "ProductCategory": "ACC",  "Division": "AS",  "InternalPrice": "10",  "Price": "20",  "Color": "Black",  "ProductGroup": "Shimano Road"  } |

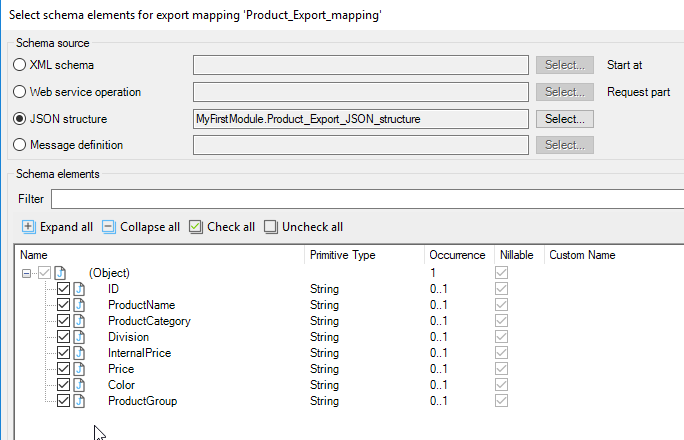
Locate the (Object) named Root…



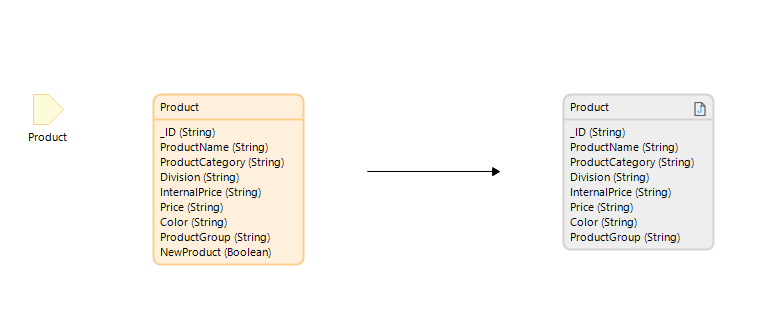
…and rename it Product.



Now create an Export Mapping in the REST folder. Configure it as shown below.

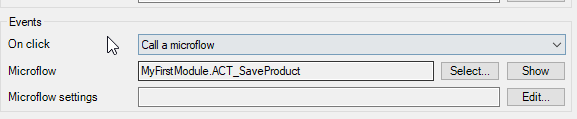


When you click Map Automatically you should see this:



## Add the Save Function

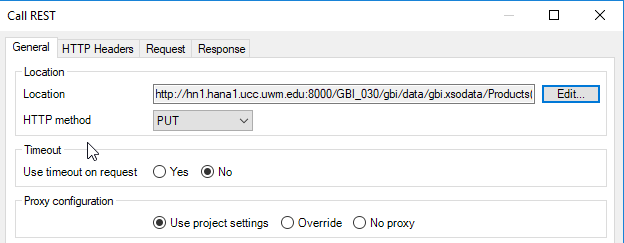
On the ProductNewEditPage, double-click the Save button and configure the Event section as shown. Choose Call a microflow for the On click event then click Select… for the Microflow. In the dialog that opens, select the Microflow folder and click the New button. Enter the name ACT\_SaveProduct.



The first activity in the microflow is a Call REST activity. A call to a server to update a record uses a PUT HTTP method. The URI will look like the one below. Note how we add the ID of the product we want to update to the end of the URI.

http://<HOST>:8000/GBI\_030/gbi/data/gbi.xsodata/Products('BOTL1000')

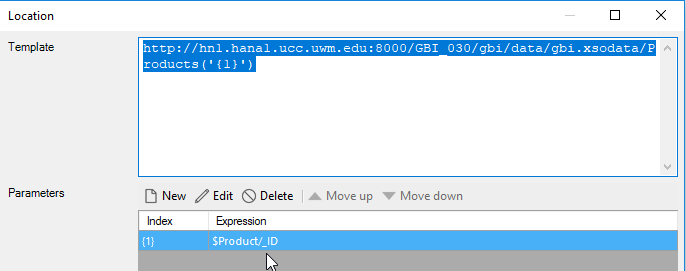
Configure the General tab of the Call REST activity as shown.



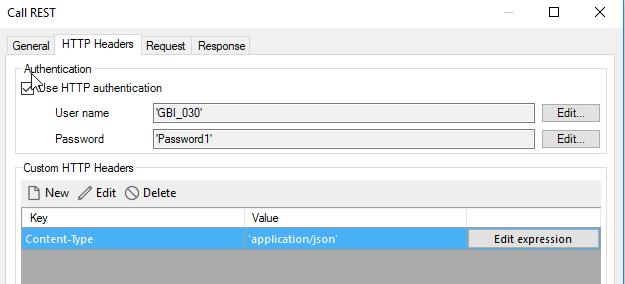
The URI to paste into the Location field is:

http://hn1.hana1.ucc.uwm.edu:8000/GBI\_030/gbi/data/gbi.xsodata/Products('{1}')

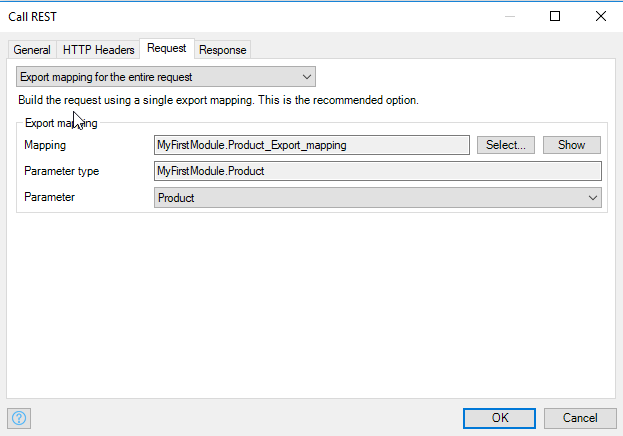
The {1} represents a parameter where Mendix will substitute a value when the microflow runs. We want to substitute the selected Product ID. The Product is a parameter in the microflow so the value to substitute is $Product/\_ID. The image below shows the configuration.



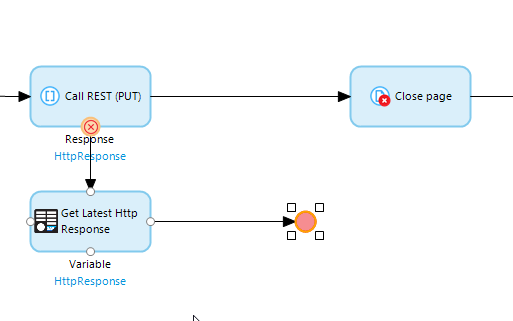
Configure the same username and password on the HTTP Headers tab that was used for the GET activity (your SAP HANA username and password). We also must add a header so that the SAP HANA system will know that the data we are sending is in JSON format. Add the Content-Type header and set the value to application/json as shown below.



The Request tab is configured using the Export Mapping we created as shown below. The Product parameter passed to the microflow will be copied using the Export Mapping into the payload sent to the server.

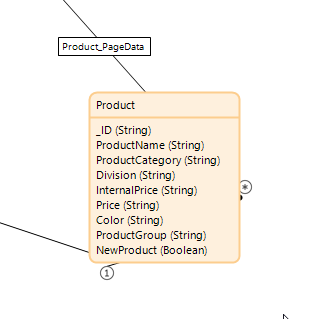


Add a Close page activity after the Call REST activity. The microflow looks like this (the image shows the Call Rest activity with error handling configured):



You should be able to edit and save a Product now. However, there is one problem with the app as it stands. If you edit the ID, you will receive an error because when the PUT request is sent, the ID attached to the URI doesn’t exist in the database. What we need to do is disable the ID field but that will cause a problem when we add the create function. The way we will handle this is to add a field to the Product entity that will be false for new products and true for existing products. We can then use this value to conditionally disable the ID widget on the ProductNewEditPage.

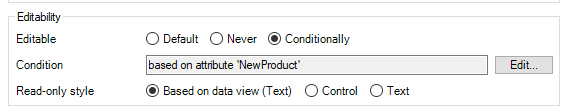
Open the Domain model and double-click the Product entity.



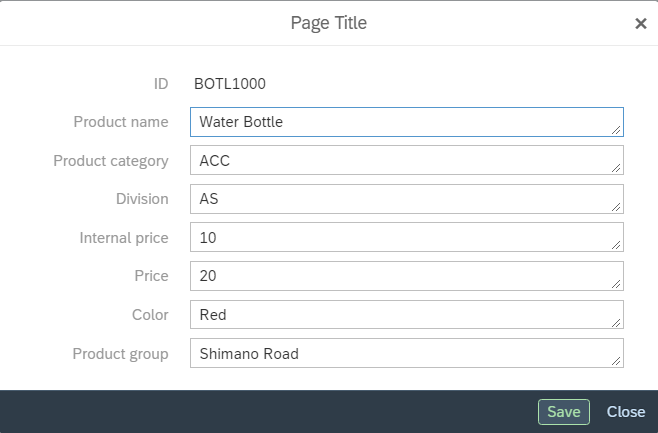
Add a Boolean field called NewProduct and set the default value to false.



Open the ProductNewEditPage and open the properties of the \_ID widget. Configure Editability as shown below.



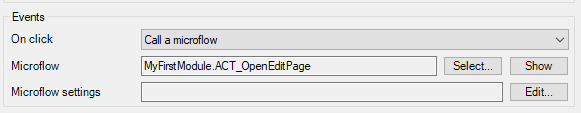
Now the ID field is disabled.



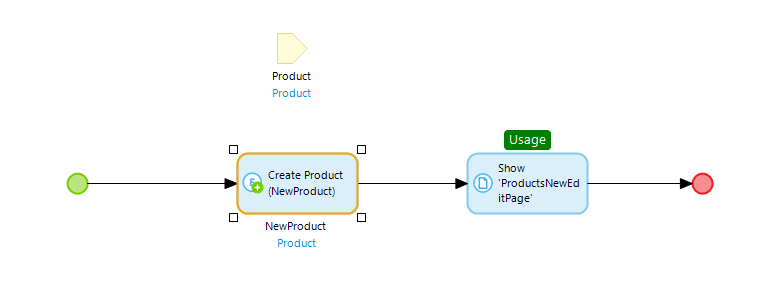
## Add the Create Function

To create a new product you send a POST request to the server. We can use the same Export Mapping that we created earlier, and we’ll edit the existing save product microflow to add the POST request.

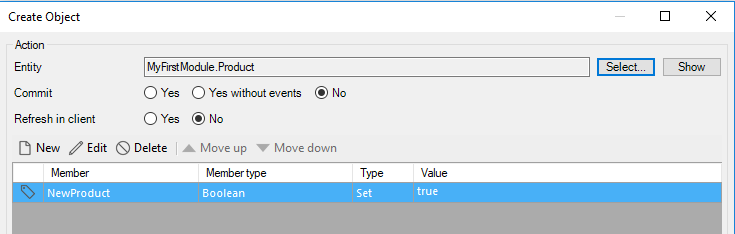
Open the ProductsPage and edit the properties of the New button so that it opens a microflow called ACT\_OpenEditPage.



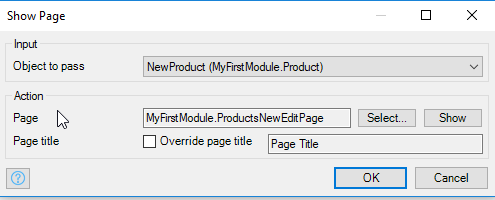
This microflow will create an empty Product object and then open the ProductNewEditPage. The microflow is shown below.



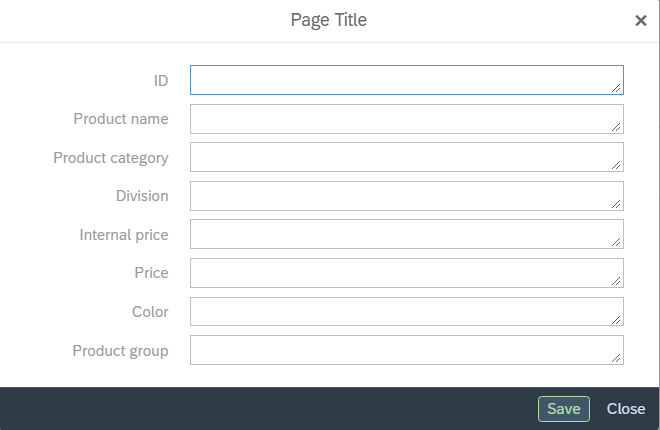
The Create Object activity creates a new Product object and sets the newProduct field to true.



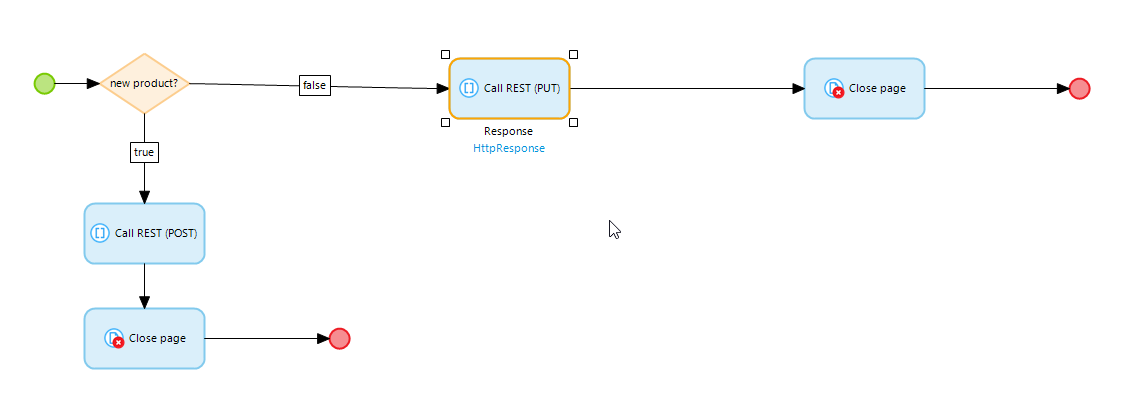
The Show Page activity opens the ProductsNewEditPage and passes the NewProduct object to it.



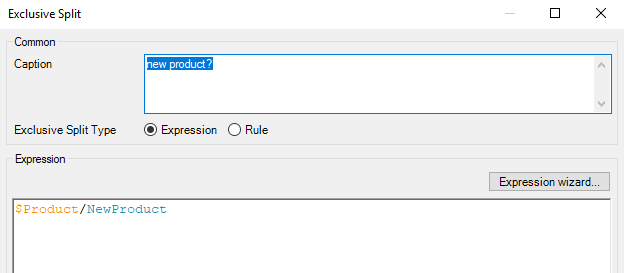
Now when you click New, the dialog opens and the ID field is editable.



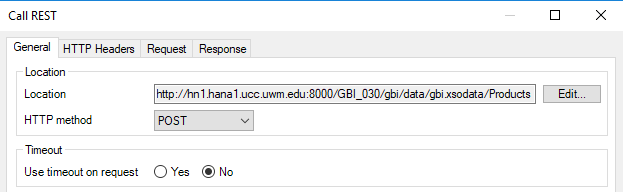
To save the new Product we must adjust the ACT\_SaveProduct microflow. The finished microflow is shown below (I’ve removed error handling activities to clarify).



The Exclusive split at the beginning of the microflow checks to see whether the Product is new. The configuration is shown below.



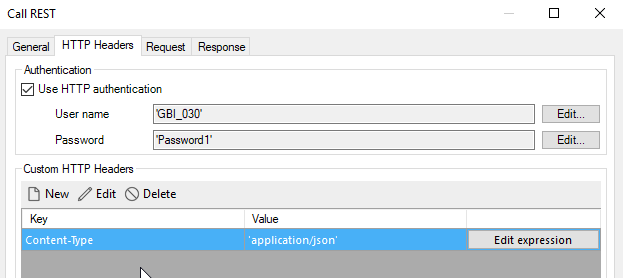
The General tab for the Call REST activity for the POST call is shown below.

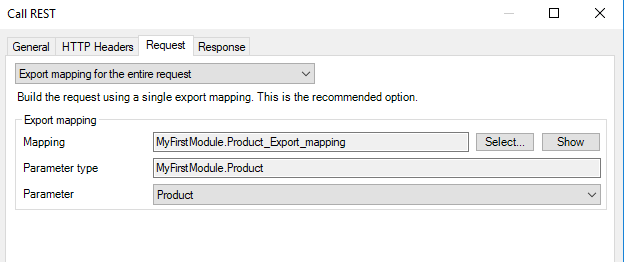


The URI for the Location field is:

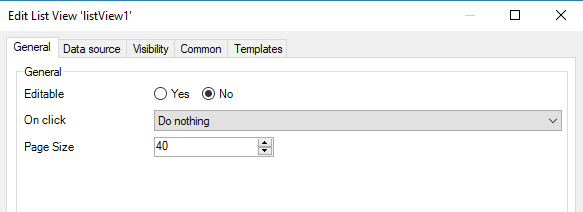
http://<HOST>:8000/GBI\_030/gbi/data/gbi.xsodata/Products

The HTTP Headers and Request tabs are the same as for the PUT request.



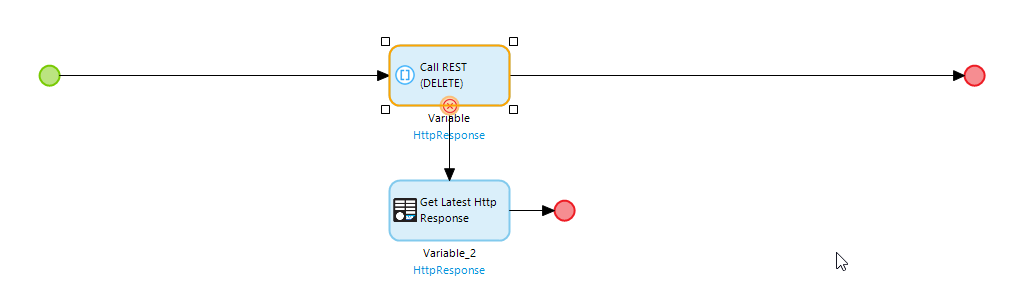


You should be able to create Products now. One change you will need to make is to increase the number of products the Listview widget shows. By default, it only shows 10. We could implement the logic for the Show more button at the bottom of the Listview but it’s easier to increase the number of products shown. Open the properties of the Listview and increase the Page size.

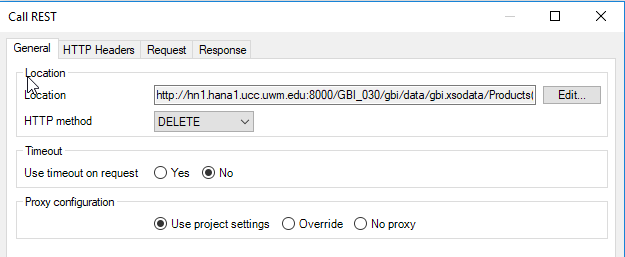


## Add the Delete Function

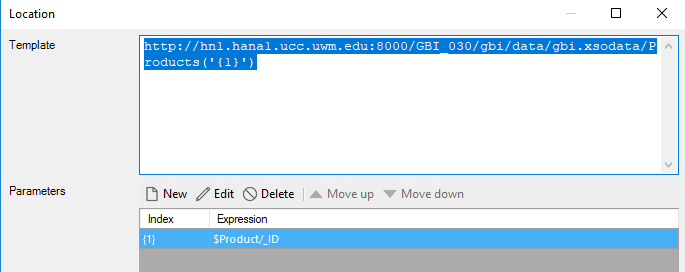
Configure the Delete button on the ProductsPage to call a microflow called ACT\_DeleteProduct. The microflow is shown below (shown with error handling).



The General tab of the Call REST activity looks like this:



The Location field is configured like the PUT request.



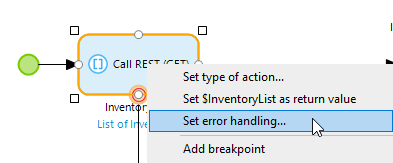
The HTTP Headers tab is configured like this and you don’t need to make any changes to the Request and Response tabs.



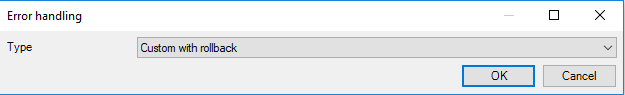
The app now has the full CRUD (Create, Read, Update and Delete) functionality. It’s still missing confirmation dialogs, validation and other niceties but I’ll leave that as an exercise.

## Appendix: Adding Error Handling to REST Calls

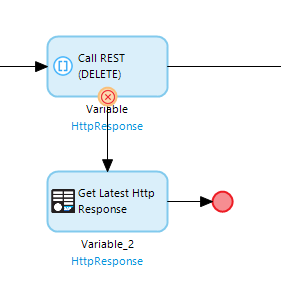
To diagnose errors with REST calls you must add error handling. This appendix describes how to do that. The first step is the right-click the Call REST activity and select Set error handling…



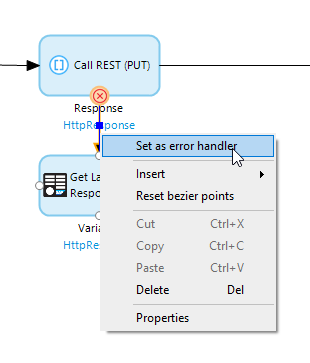
Choose Custom with rollback.



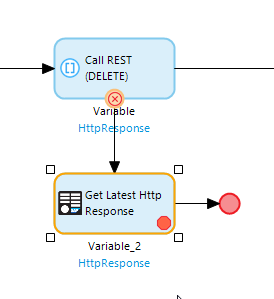
Next, drag down from the border of the Call REST activity and add an activity. Configure the activity as a Get latest HTTP response. Next drag out from this activity and add an End event.



Right-click the line between the Call REST activity and the Get Latest Http Response activity and select Set as error handler.



If you are getting errors in a REST call, right click the Get Latest HTTP Response activity and add a breakpoint. The red dot indicates an activate breakpoint.



In the app perform the function that will invoke the microflow and the microflow will pause if it gets to the breakpoint. You can then look at the Variables tab in the Mendix Modeler and exam the error messages.