# Unique attributes

## Pattern #2

A very common requirement in applications is for an attribute to be unique. A unique attribute (or set of attributes) means that no two records of an entity can have the same value for the attribute (or set of attributes).

The first part of this exercise requires you to create an entity, define one of its attributes as unique and guarantee that you cannot create two records with the same value for the unique attribute. In the second part of the exercise you will establish a many-to-many relationship with another entity and guarantee that you cannot have more than one relationship between the same two records of the different entities.

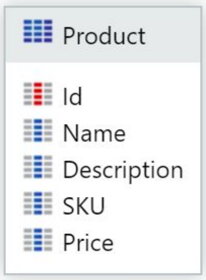
KEYWORDS: **data-model**, **entity**, **attribute**, **index**, **unique**, **many-to-many**, **relationship**, **master-detail**, **junction**, **scaffolding**, **user exception**

## Develop the MyOrders application

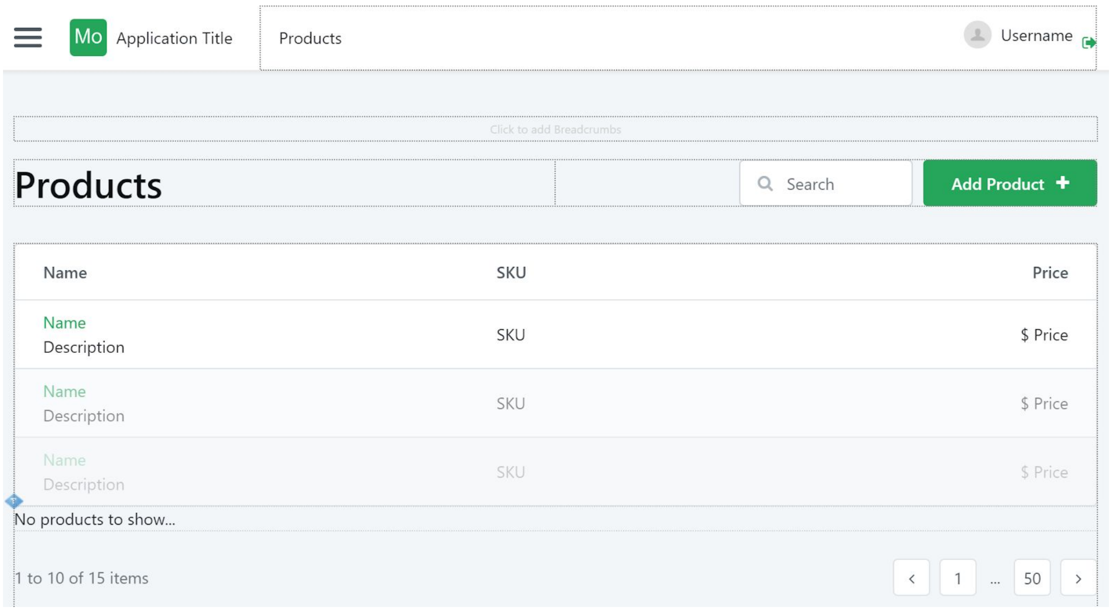
The MyOrders application allows users to build a catalogue of products and add them to orders. The two main concepts are a **product** and an **order**. When going through the catalog, the application will allow the user to list the products, and add new ones. Products have a unique SKU, and when adding them to an order the user will be able to ordered several at a time

### Part I - Creating the base data-model and application screens

1. Create a new application called **MyOrders_<*your initials*>** and add a Reactive Module with the same name
2. Create the **Product** entity entity, with the following attributes:
   a) A mandatory *Text* attribute named **Name**, that can hold up to **50** characters
   b) A *Text* attribute named **Description**, with **160** characters in length
   c) An mandatory *Text* attribute named **SKU**, 14 characters long
   d) A mandatory *Currency* attribute named **Price**



3. Use scaffolding to create a screen that lists all products, connected to another screen that shows the details of a single product. For better usability, make sure that
    a) in the listing screen, the description shows below the Name attribute and both the menu entry and the title show as **Products**, and

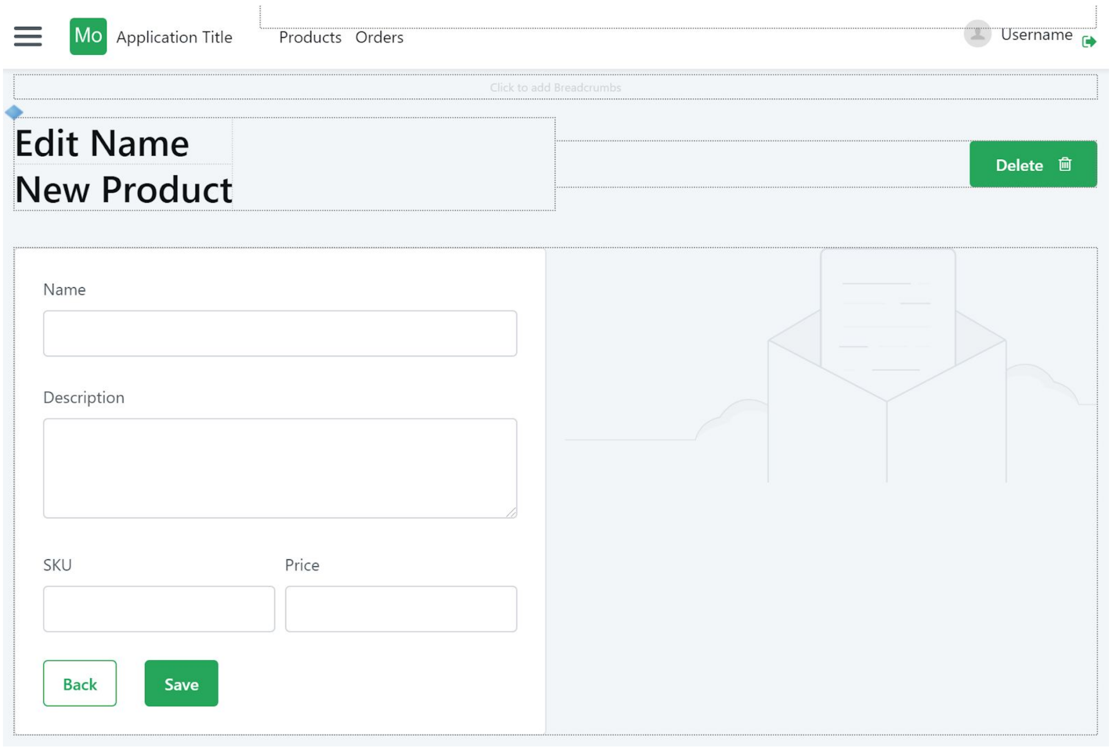


| Name | SKU | Price |
|---|---|---|
| Name<br>Description | SKU | $ Price |
| Name<br>Description | SKU | $ Price |
| Name<br>Description | SKU | $ Price |

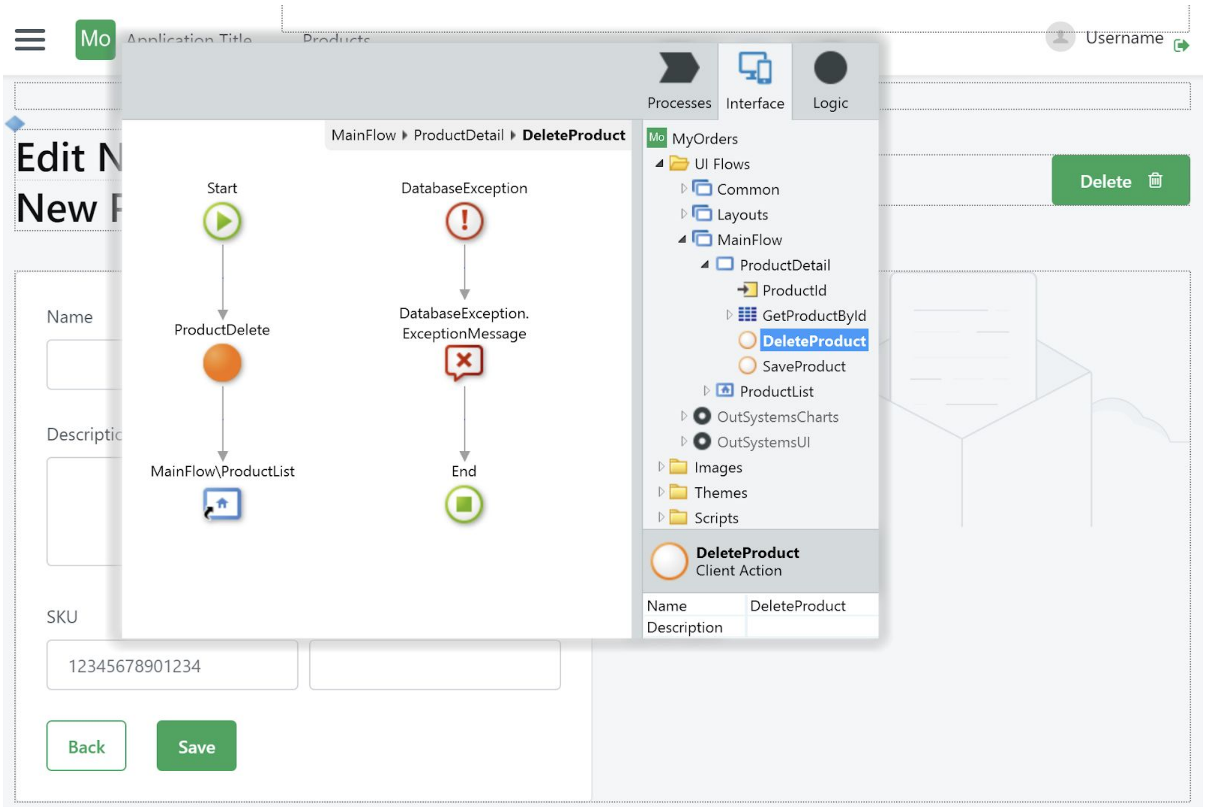
No products to show...

1 to 10 of 15 items

    b) on the detail screen, description should be a multi-line input and the screen title should be **Edit *name-of-product*** (or **New Product** when adding products)



4. Add the Delete functionality to your detail screen
    a) Define a new server action **ProductDelete** that receives a *Product Identifier* and uses the DeleteProduct entity action to delete the Product from the database
    b) Add a button to the *Actions* placeholder, and assign it to a screen action that calls the *ProductDelete* server action



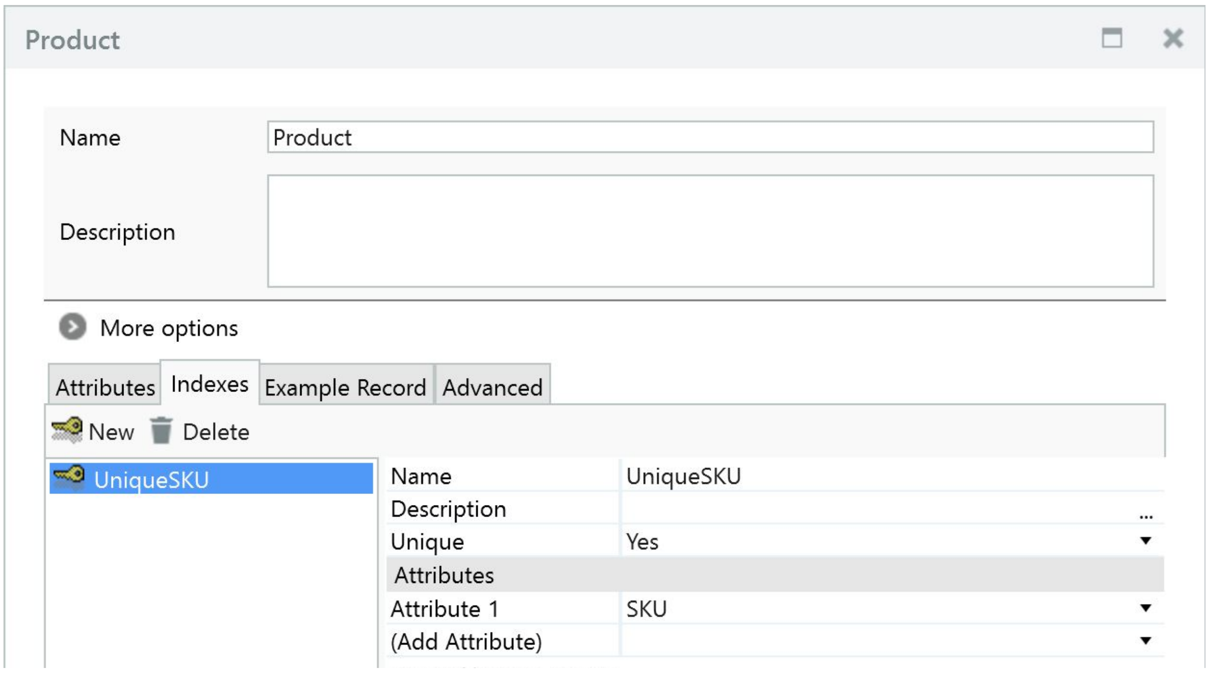
5. Publish your module and use it to add two new products with the same SKU

QUESTION: Did you manage to add both products? How can you guarantee that the application doesn't allow you to add the second product?

CONTINUES NEXT PAGE

6. Guarantee that the SKU attribute is unique

   a) Open the **Product** entity details window and add an unique index to the SKU attribute



Product

| | |
|---|---|
| Name | Product |
| Description | |

More options

| | |
|---|---|
| Name | UniqueSKU |
| Description | |
| Unique | Yes |
| Attributes | |
| Attribute 1 | SKU |
| (Add Attribute) | |

7. Publish your module and use it to add two new products with the same SKU

QUESTION: Did you manage to add both products? How can you make the message displayed user friendly?

CONTINUES NEXT PAGE

8. Add a check on the **ProductCreateOrUpdate** server action to guarantee that there are no Products already with the SKU

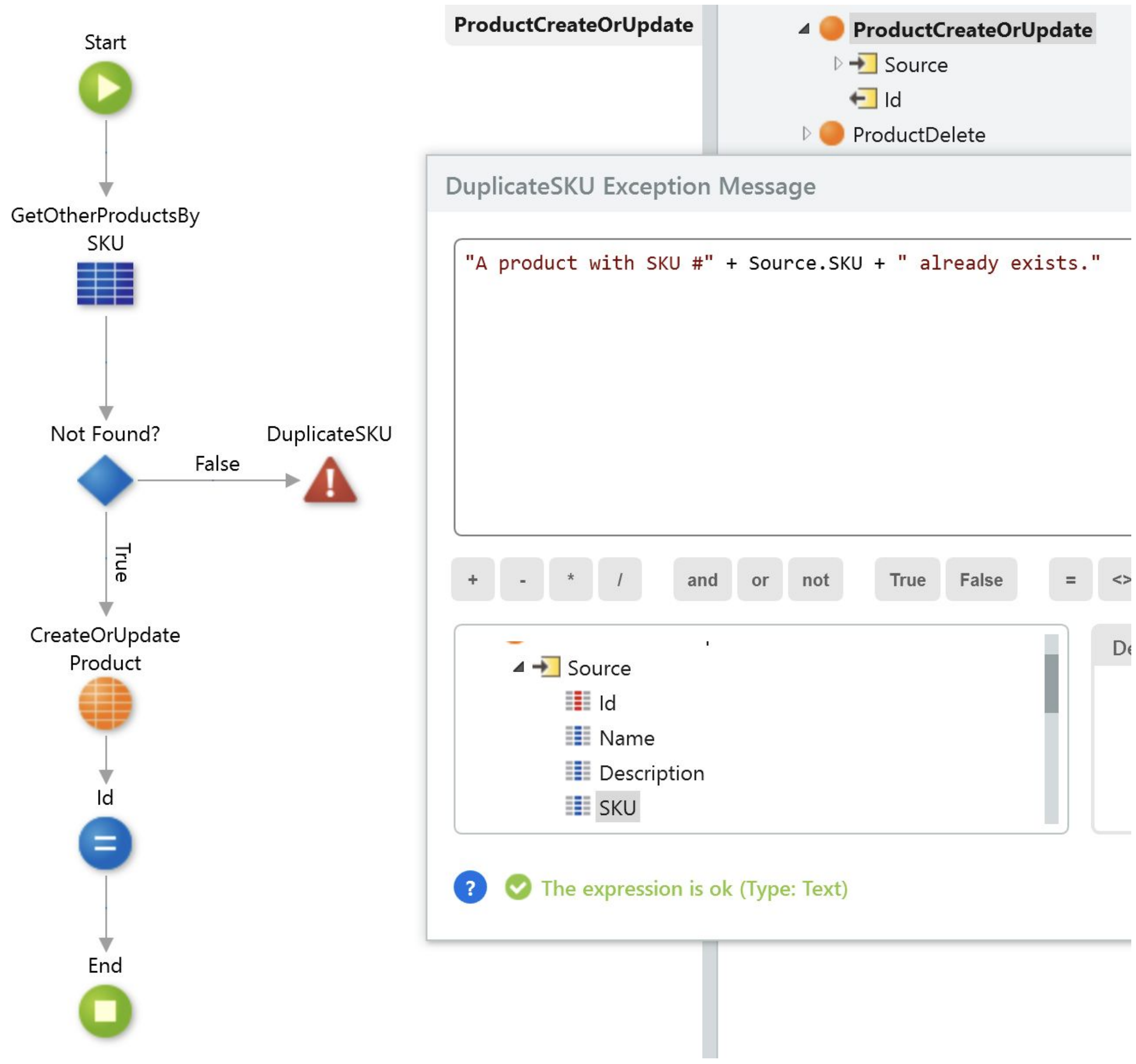
   a) At the beginning of the **ProductCreateOrUpdate** server action, add an Aggregate to fetch products that have the same SKU as the one being created/updated

   ---

   NOTE: Make sure you filter out the **Product** being updated from the results of the Aggregate

   ---

   b) Check if there were any products returned and if so throw a Custom User Exception

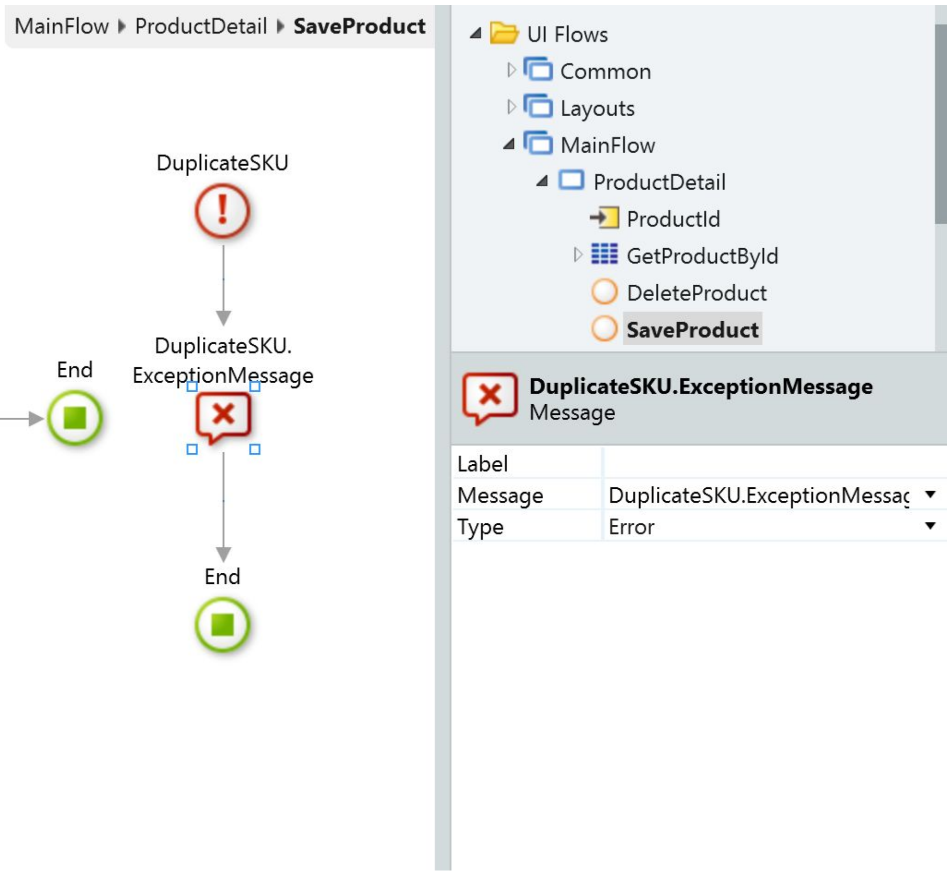


```
"A product with SKU #" + Source.SKU + " already exists."
```

The expression is ok (Type: Text)

CONTINUES NEXT PAGE

9. On the **SaveProduct** screen action, add an Exception Handler for the Custom User Exception that shows the exception message content in an error feedback message



**DuplicateSKU.ExceptionMessage**
Message

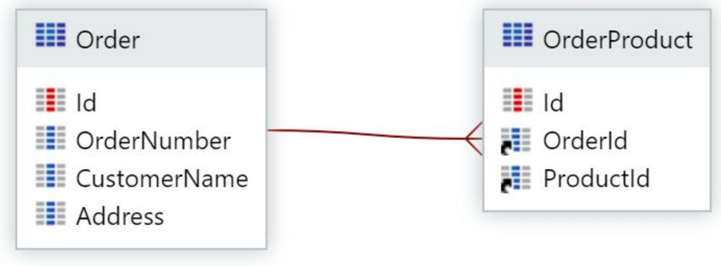
| | |
|---|---|
| Label | |
| Message | DuplicateSKU.ExceptionMessag |
| Type | Error |

10. Publish your module and use it to add two new products with the same SKU. Notice how now you get a user friendly feedback message

END OF PART I

## PART II - Extending the data-model with a many-to-many relationship

1. Create the **Order** entity, and the **OrderProduct** junction entity

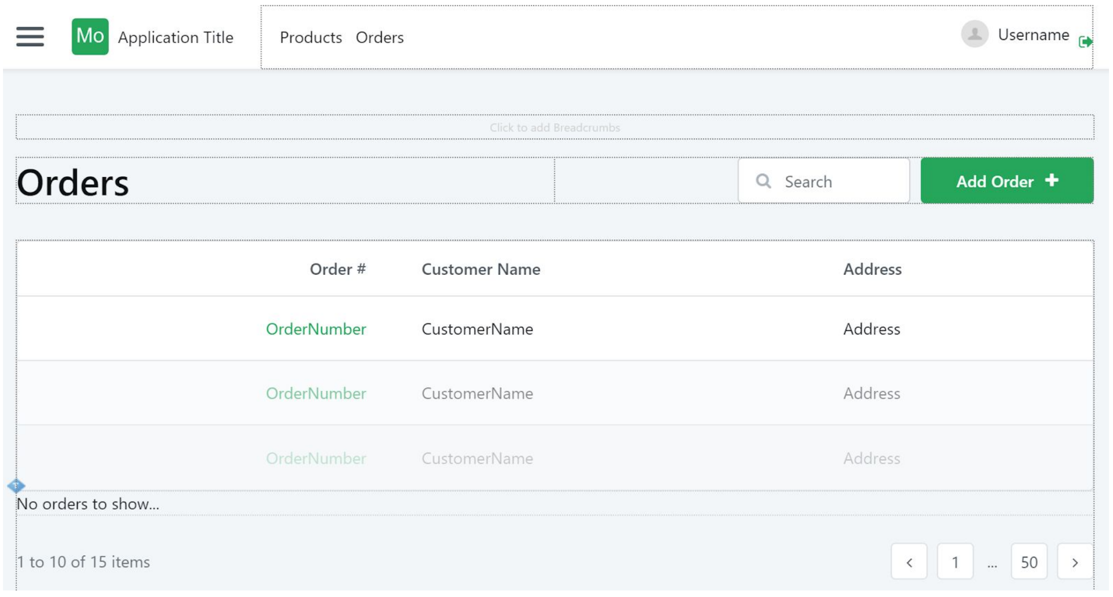


   a) Start by creating the **Order** entity, with the following attributes:
      - A mandatory *Integer* attribute named **OrderNumber**
      - A *Text* attribute named **CustomerName**, with a length of **50** characters
      - A mandatory Text attribute named **Address** of up to **200** characters

   b) Then create the **OrderProduct** junction entity, with the following attributes:
      - A mandatory *Order Identifier* attribute named **OrderId**
      - A mandatory *Product Identifier* attribute named **ProductId**

2. Use scaffolding to create a screen that lists all orders, connected to another screen that shows the details of a single order. For better usability, make sure that:

   a) in the listing screen, the first column shows the Order number and links to the detail screen, and both the menu entry and the title show as **Orders**, and



Orders

| Order # | Customer Name | Address |
|---|---|---|
| OrderNumber | CustomerName | Address |
| OrderNumber | CustomerName | Address |
| OrderNumber | CustomerName | Address |

No orders to show...

1 to 10 of 15 items

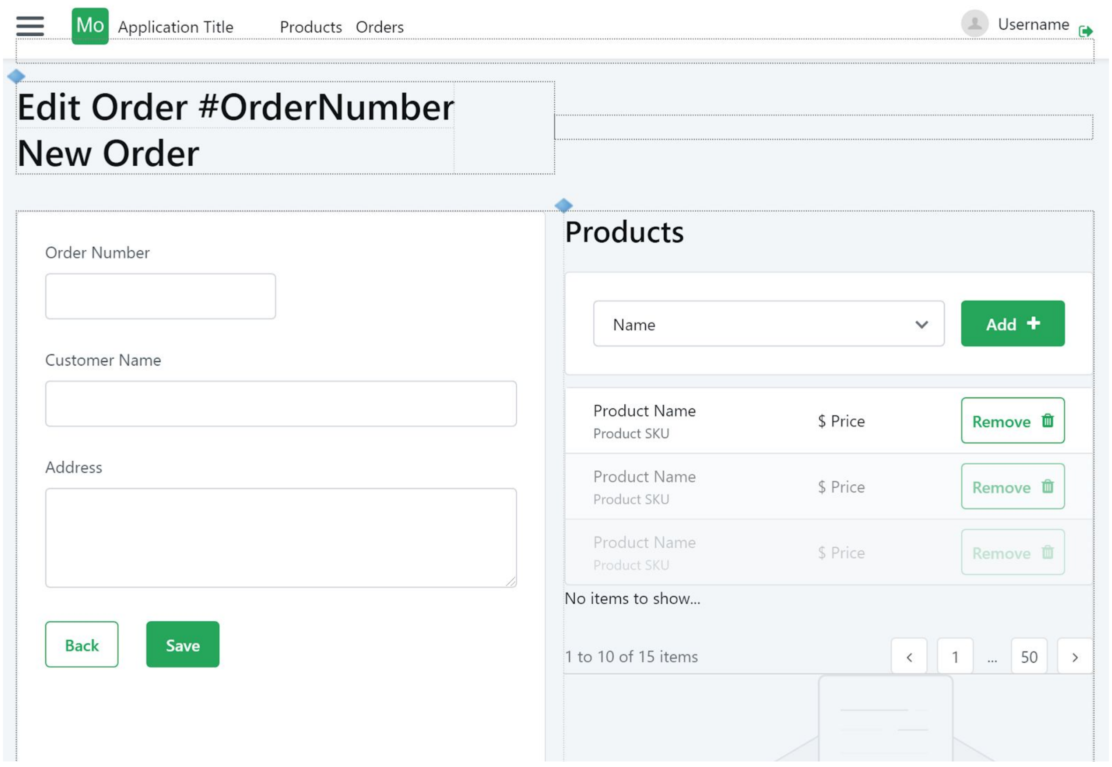
CONTINUES NEXT PAGE

b) on the detail screen, the address should be a multi-line input and the screen title should be **Edit Order *#order-number*** (or **New Order** when adding orders).
For orders already created, it should show on the second column the list of products already in the order, with the option to add new ones and remove existing ones.

---

NOTE: For both the Add and Remove buttons, you will need to call the respective wrapper Server Actions, clear the inputs so they are ready to add the next product and end with a refresh of the Aggregate that fetches the order's products

---



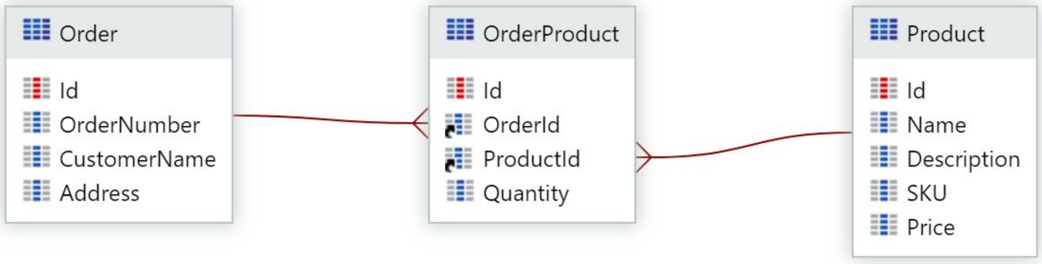
3. Publish your module and use it to create an order and add the same product to it twice

QUESTION: How can we modify our data-model so the user cannot add the same product twice to the same order, and instead specify a quantity?

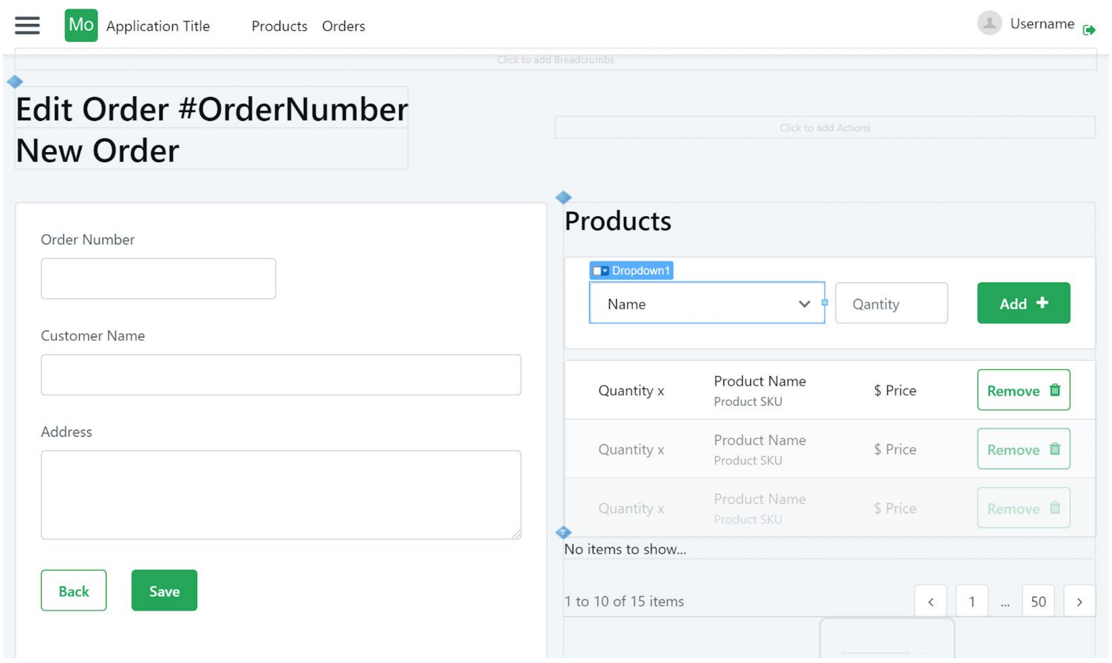
CONTINUES NEXT PAGE

4. Add a qualifier to the relationship between Order and Product, and restrict it to not allow the same relationship more than once

   a) Add a mandatory *Integer* attribute named **Quantity** to the **OrderProduct** entity.



   b) Open the **OrderProduct** entity details window and add an unique index with both the **OrderId** and **ProductId** attributes.

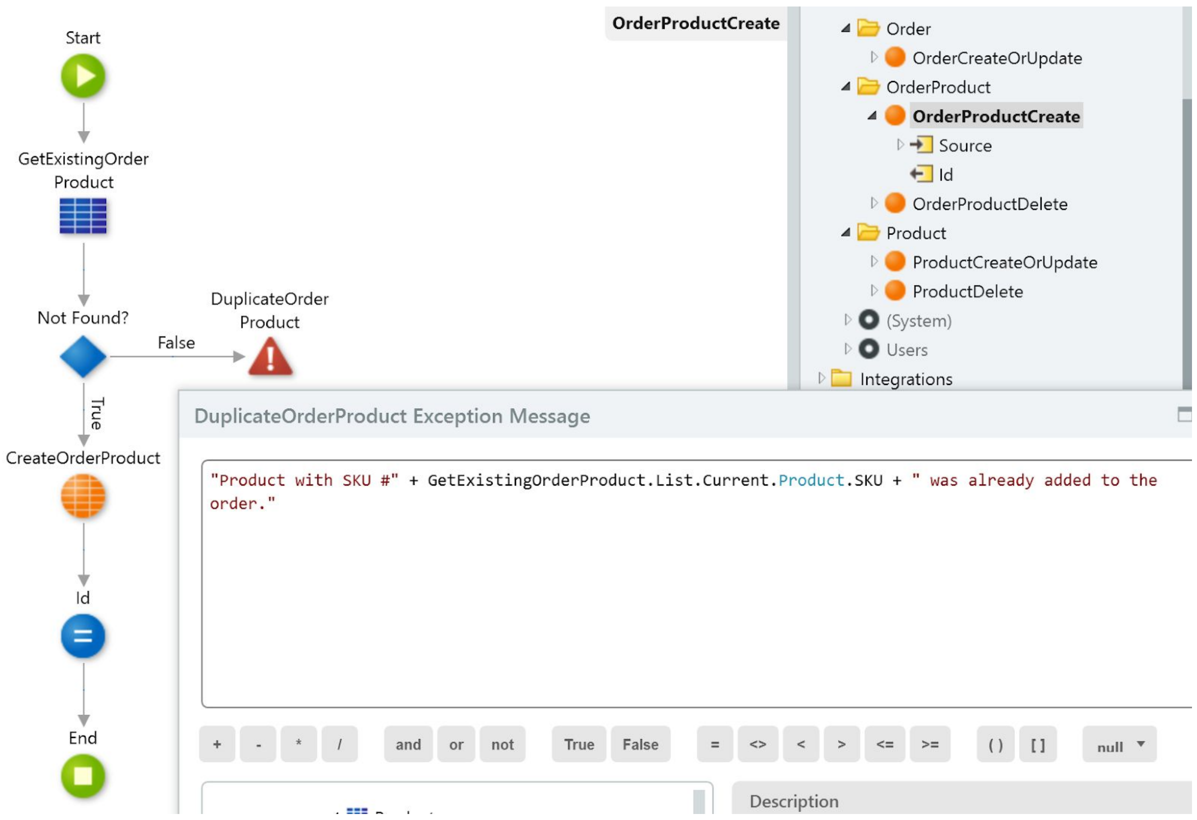


CONTINUES NEXT PAGE

5. Similar to what you've done in Part I, add a check on the **OrderProductCreate** server action to guarantee that the **Order** doesn't already have the **Product** being added

    a) At the beginning of the **OrderProductCreate**, add an Aggregate to fetch **OrderProducts** that have the same **OrderId** and **ProductId** as the one being created

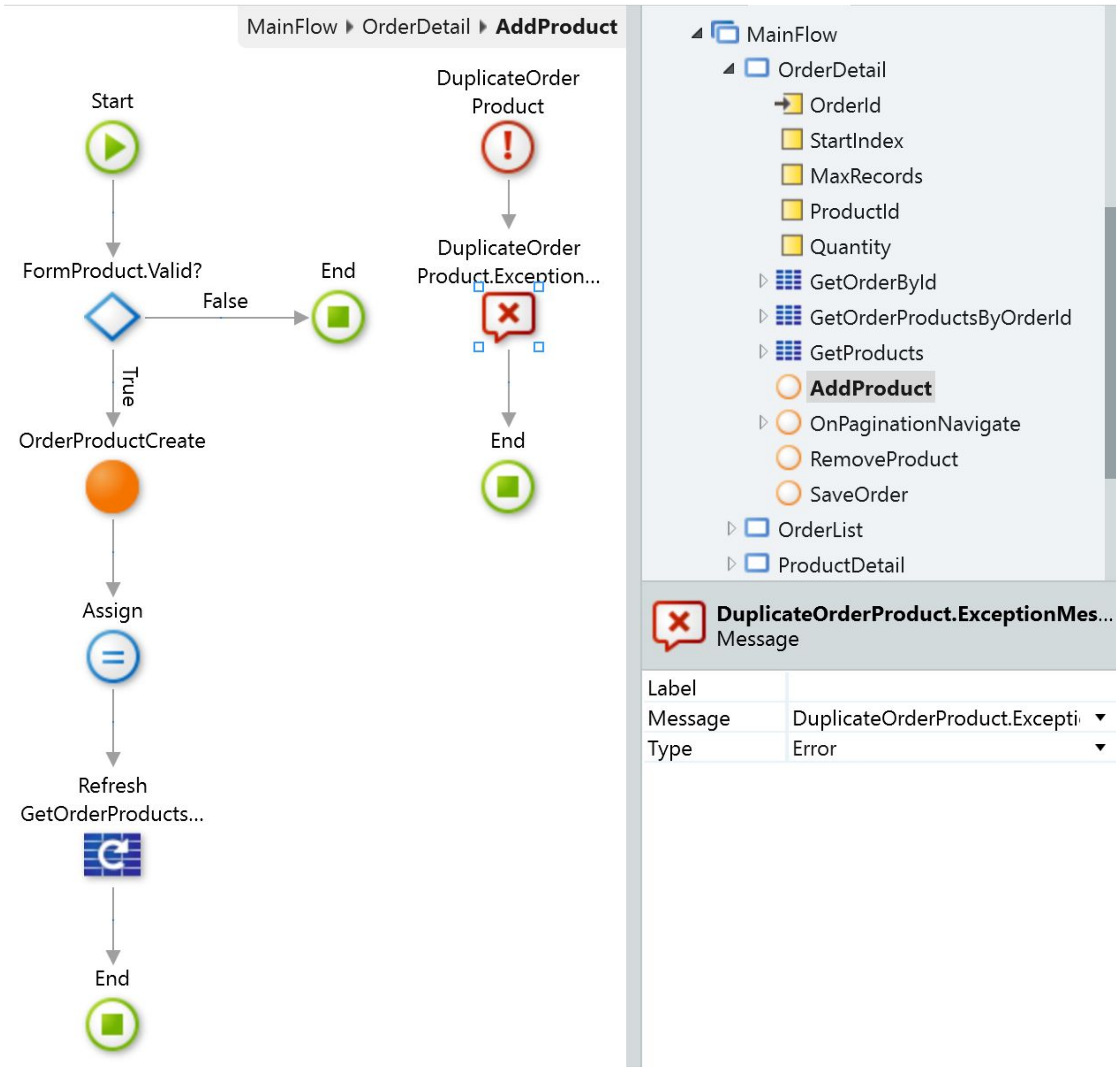
    b) Check if there were any **OrderProducts** returned and if so throw a Custom User Exception



```
"Product with SKU #" + GetExistingOrderProduct.List.Current.Product.SKU + " was already added to the
order."
```

CONTINUES NEXT PAGE

6. On the **SaveProduct** screen action, add an Exception Handler for the Custom User Exception that shows the exception message content in an error feedback message



**DuplicateOrderProduct.ExceptionMes...**
Message

| | |
|---|---|
| Label | |
| Message | DuplicateOrderProduct.Excepti |
| Type | Error |

7. Publish your module and use it to create an order and add the same product to it twice. Notice how now you get a user friendly feedback message

END OF PART II