

Uploading an Employee Picture

Table of Contents

Outline.....	2
Scenario.....	3
Employee Detail Screen	3
How-To.....	4
Getting Started	4
Creating a New Entity	4
Fetching the Picture from the Database	7
Uploading the Picture	10
Saving the Employee Picture to the Database	14
Explore the Logic	14
Create the Logic to Save the Picture in the Database	18
Finish the Logic and Test the App	23
Customizing the Layout	24
Wrapping Up.....	34
References	34

Outline

In this tutorial, you will extend the Employee Directory application, with a new functionality that will allow users to **upload a picture of an employee**, in the EmployeeDetail Screen.

To implement this new functionality, you will start by extending the data model to save the employee picture, and relate the picture with the employee, where an employee can only have one picture and a picture only belongs to one employee.

Last but not least, you will modify the EmployeeDetail Screen to allow uploading a picture, and then displaying it.

Scenario


The Employee Directory App at this point should have two Screens: one to list employees and one to edit employee information.

In this tutorial, the application will allow users to add a picture to an employee in the Employee Detail Screen.

Employee Detail Screen

The Employee Detail Screen displays all information of an employee stored in the database. A user should be directed to this Screen when they click on the name of an employee, or when they select the option to add an employee, on the Employee List Screen.

Besides viewing the employee information, it is also possible to modify the data and save the changes to the Database by clicking on the Save Button. Now, we want to be able to upload a picture for each Employee in their respective Employee Detail Screen and also save it to the Database.

 EmployeeDirectory Employees
 Login

Edit Employee

Name *

Birth Date *

Email *


Phone *

Job Position *

Hiring Date *

Department

Location



Change

Bio

Top-ranked sales manager, contributed to record sales and new account development.
Proven ability to lead sales teams to achieve established targets.

Is Active

☒

How-To

In this section, we'll show a thorough step-by-step description of how to implement the scenario described in the previous section.

Getting Started

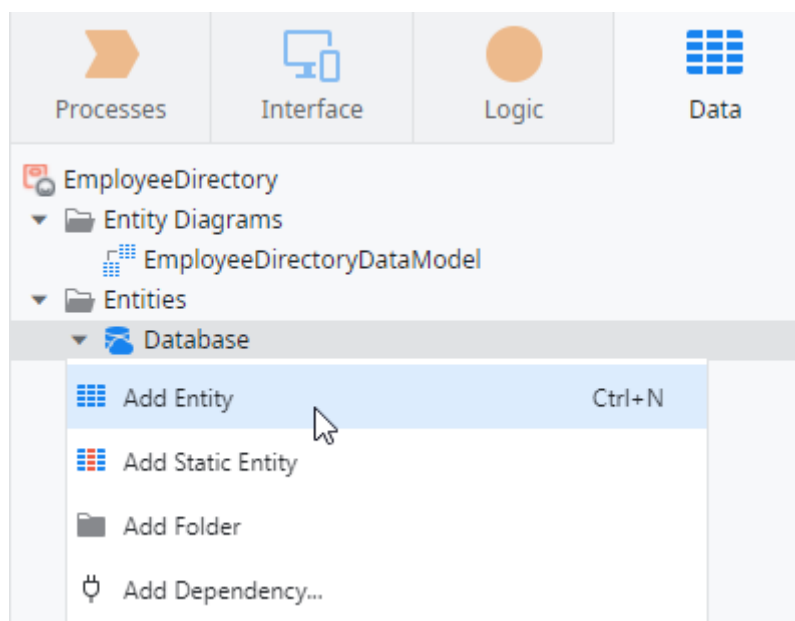
In this tutorial we are assuming that you have already created two screens with Service Studio accelerators: the **Employees** and **EmployeeDetail** Screens.

In case you haven't yet created the Employees and EmployeeDetail Screens yet, it is important to go back to the previous tutorial, and then come back to continue with the employee picture. Otherwise, just install the Quickstart application available in the Lesson Materials.

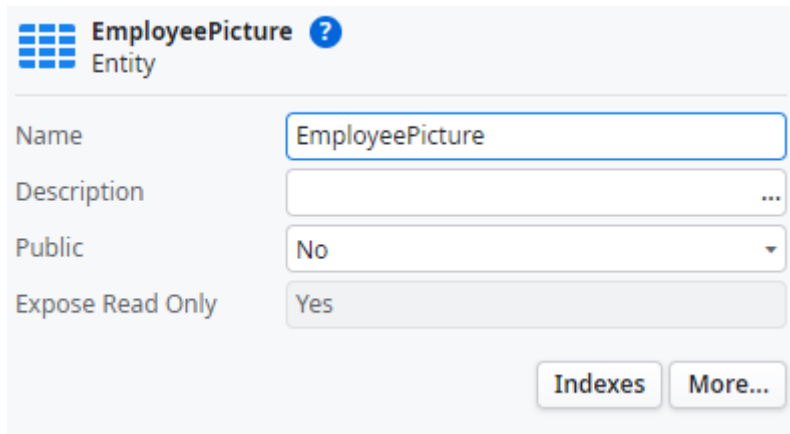
Creating a New Entity

Let's now go back to Service Studio and the **EmployeeDirectory** module we are working on, and let's start by creating the Entity that will hold the employee pictures in the database.

- 1) Select the **Data** tab on the top right corner, right-click the Database and select **Add Entity**

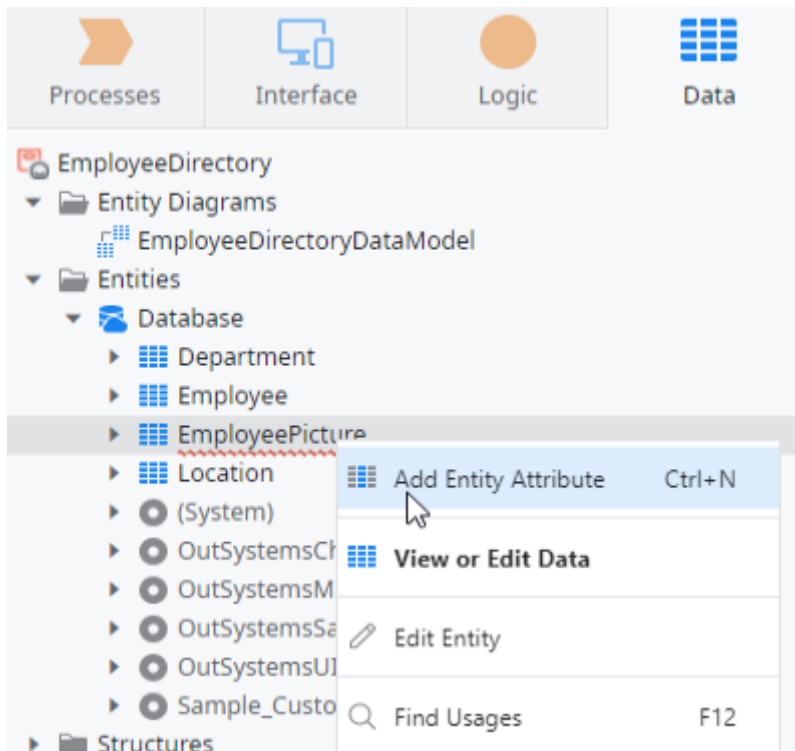


- 2) Name the new Entity as *EmployeePicture*.



The screenshot shows the configuration form for a new entity named "EmployeePicture". The form has a title bar with a grid icon, the name "EmployeePicture", and a question mark icon. Below the title bar, there are four input fields: "Name" (containing "EmployeePicture"), "Description" (empty), "Public" (set to "No"), and "Expose Read Only" (set to "Yes"). At the bottom right, there are two buttons: "Indexes" and "More...".

- 3) Now, it's time to add the attributes of the Entity. Right-click the newly created entity, and select **Add Entity Attribute**.



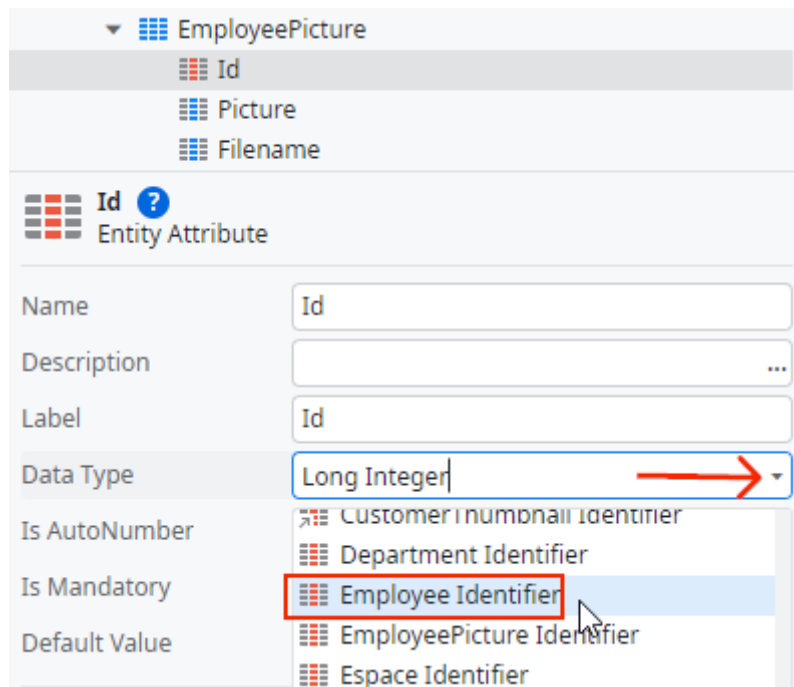
- 4) Name the attribute as *Picture* and confirm its Data Type is set to **Binary Data**. This is the attribute where the actual picture will be saved.

The screenshot shows the OutSystems IDE interface. At the top, a tree view displays the 'EmployeePicture' entity with attributes 'Id' and 'Picture'. The 'Picture' attribute is selected. Below the tree, the 'Picture' attribute is detailed as an 'Entity Attribute'. A form below this shows the configuration for the attribute: 'Name' is 'Picture', 'Description' is empty, 'Label' is 'Picture', 'Data Type' is 'Binary Data' (selected from a dropdown), and 'Is Mandatory' is 'No' (selected from a dropdown).

Notice that OutSystems automatically changes the Data Type to Binary Data. This is an accelerator that guesses the Data Type of a field based on its name. However, it's always important to double-check if the Data Type is the one we want.

- 5) Repeat the previous step to create another attribute. Name it *Filename* and make sure the Data Type property is set to **Text**.
- 6) Now, we need to create the relationship between the EmployeePicture and Employee Entities. To do that, select the **Id** attribute of the EmployeePicture

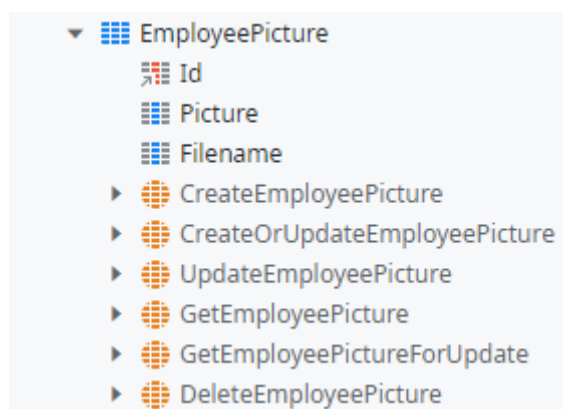
Entity, expand the Data Type dropdown, and scroll down until you find **Employee Identifier**. Select it to change the Data Type.



In this case, an Employee Picture is like an extension of the Employee Entity. So, the employee and its respective picture will share the same unique identifier in the database (their **Ids** will have the same value).

Also, notice that the **Id** attribute, to uniquely identify each element in the database (e.g. employee, picture, ...) is automatically created for you by OutSystems, but you can change any of its properties later.

Your Entity should look like this:

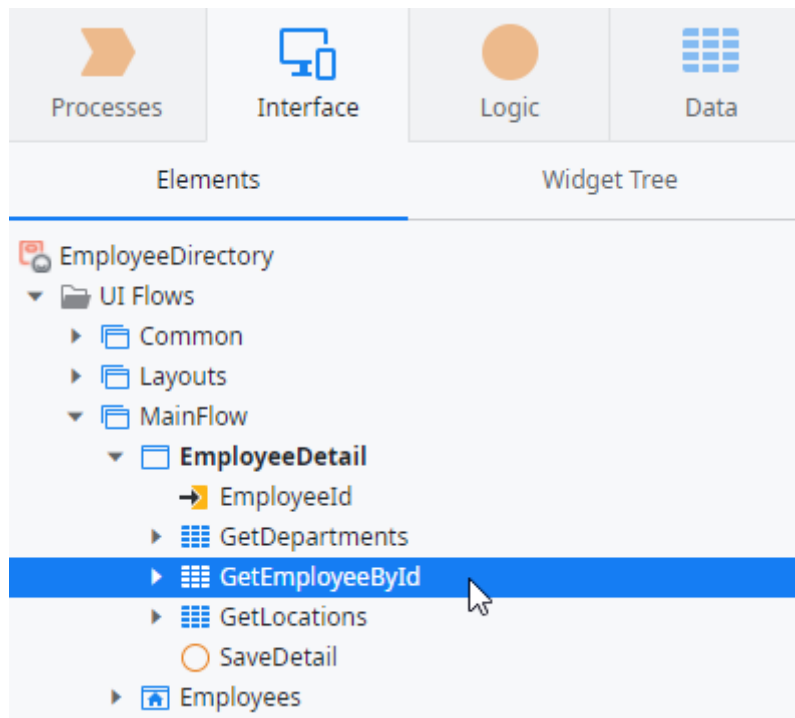


Fetching the Picture from the Database

At this point, we have adapted the data model to include the employee picture. So, now we need to change the EmployeeDetail Screen to support this new functionality. Let's

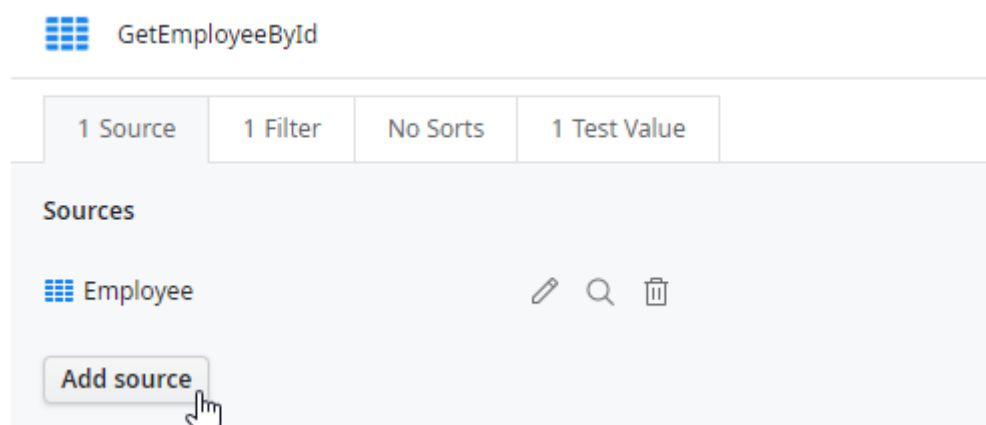
start by adjusting the data that is fetched to populate the EmployeeDetail Screen, and make it include as well the picture.

- 1) Switch to the **Interface** tab in Service Studio, expand the **EmployeeDetail** Screen on the right sidebar, and double-click the **GetEmployeeById** element to open it.

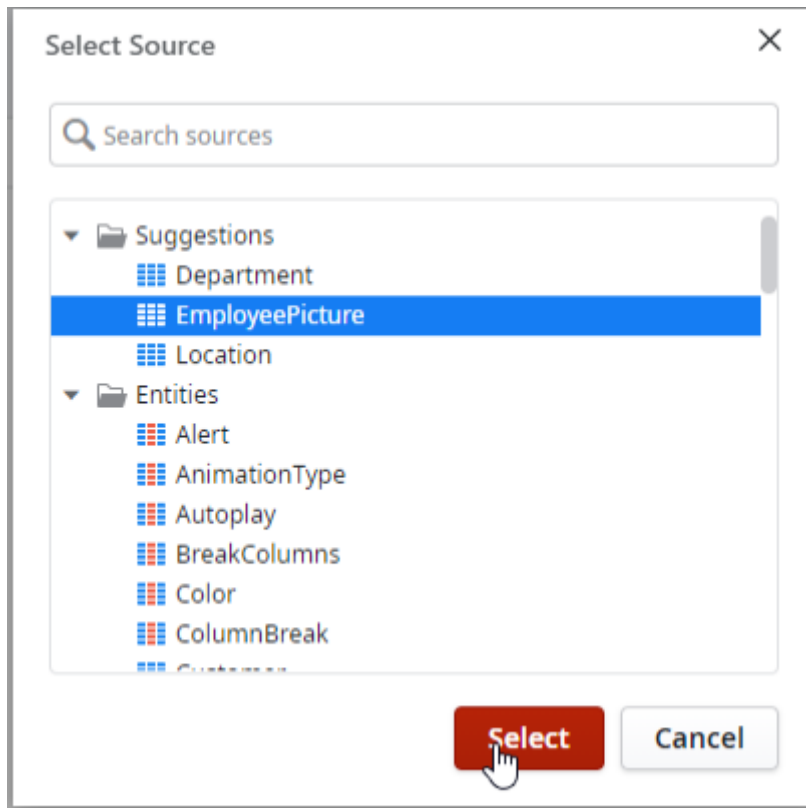


This element is called Aggregate. An Aggregate fetches data from your Entities, without requiring any knowledge of SQL. This Aggregate in specific fetches the data from the **Employee** Entity, which is then used to populate the EmployeeDetail Screen.

- 2) Now, you need to also fetch data from the EmployeePicture Entity. In the Aggregate you just opened, click on the **Add source** button.

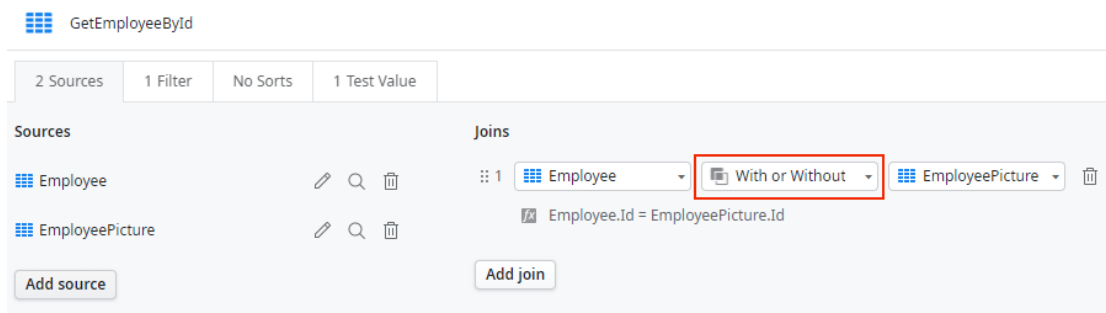


- 3) In the pop-up dialog that appears, select the **EmployeePicture** Entity and click **Select**.



The Aggregate now fetches data from two Entities: employees and their pictures. Since the Entities are related to each other, OutSystems automatically creates a Join between the Entities.

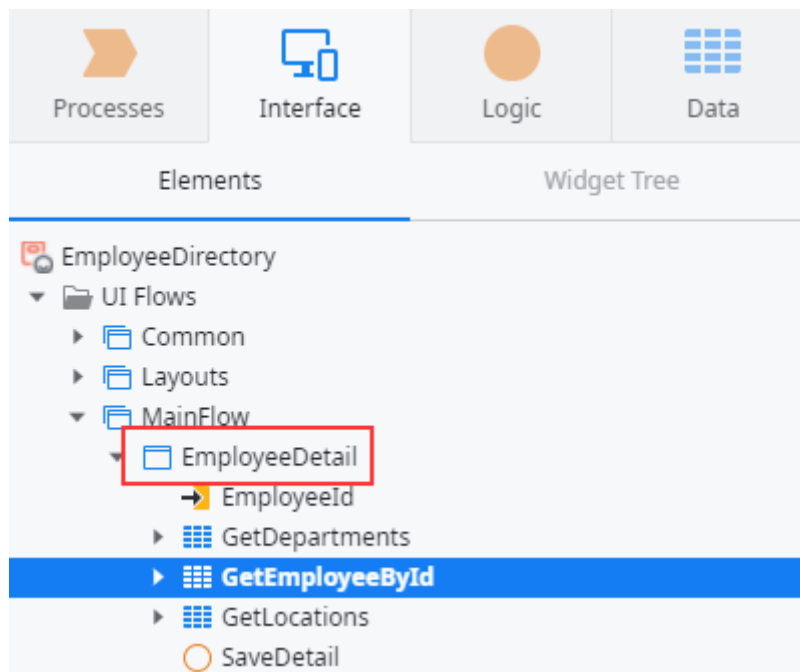
- 4) Make sure that the Join created is set to **With or Without**. This means that the employee information is fetched, regardless if they have a picture or not.



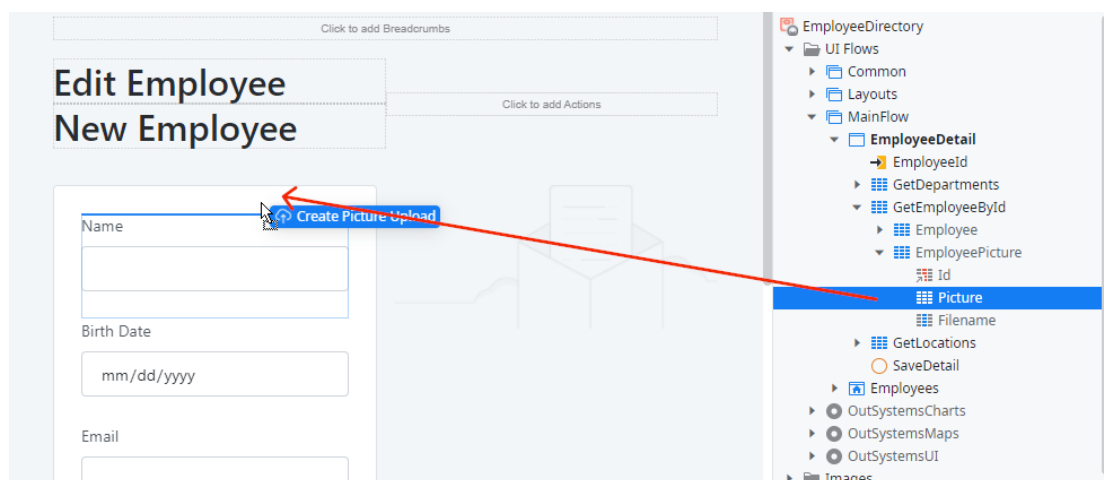
Uploading the Picture

Our Aggregate is ready, so we need to work on the visible part of the EmployeeDetail Screen. So, you need to make sure that the Form with the employee information has an additional field, that allows the user to upload a picture of the employee.

- 1) Double-click the **EmployeeDetail** Screen to open it.



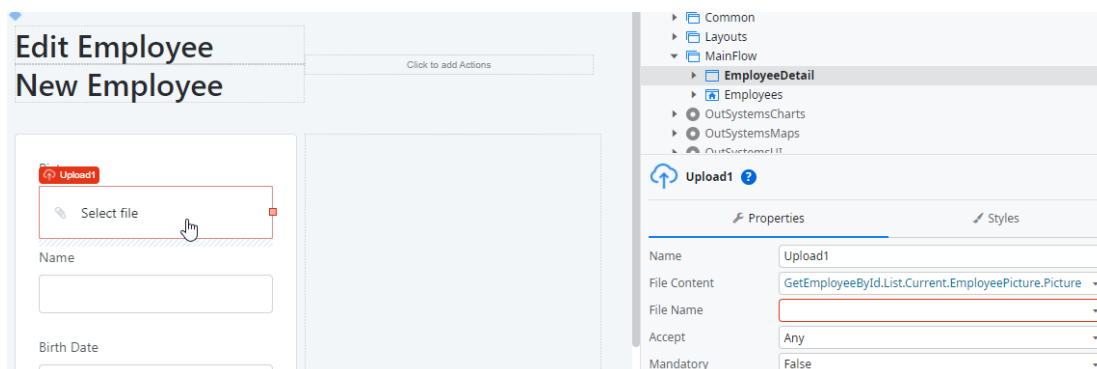
- 2) Expand the **GetEmployeeById** and the **EmployeePicture** inside it, to see the Entity's attributes. Drag the **Picture** attribute and drop it on the EmployeeDetail Screen, right above the Name input field.



This will automatically create a new element on the Screen, called Upload. This is the element that will allow a user to upload the picture in the app. Since the

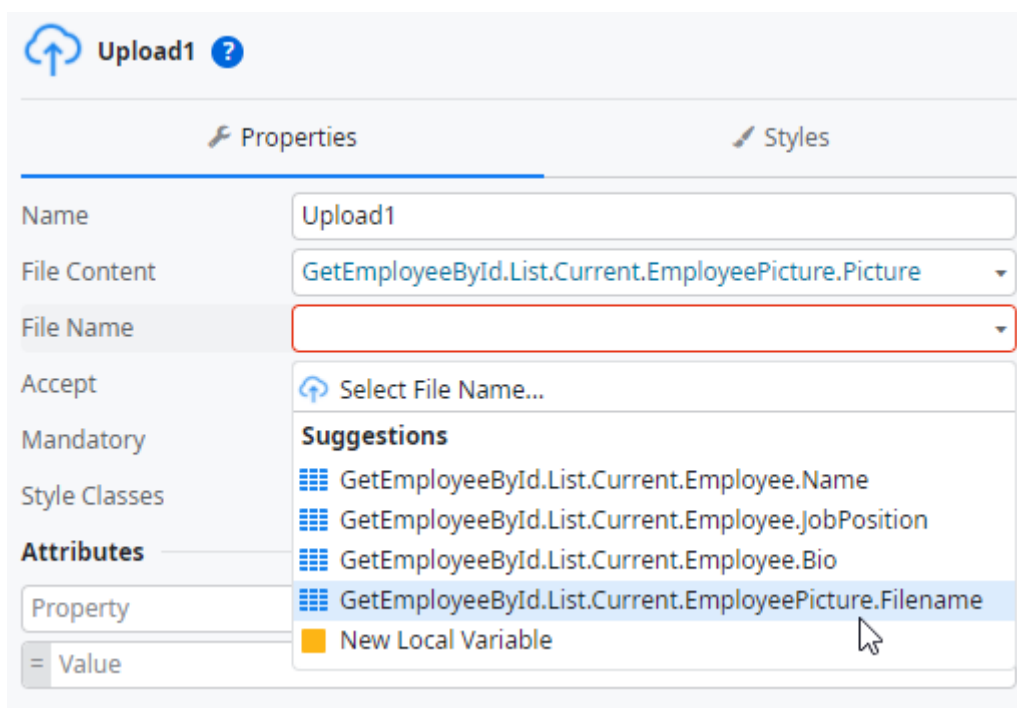
Picture attribute is a binary data, OutSystems automatically creates an Upload element, when the attribute is dropped on the Screen.

- Now we have an error in the module. Select the **Upload** element that you just added. If you look to the right side of Service Studio, you will find the properties of the Upload element.



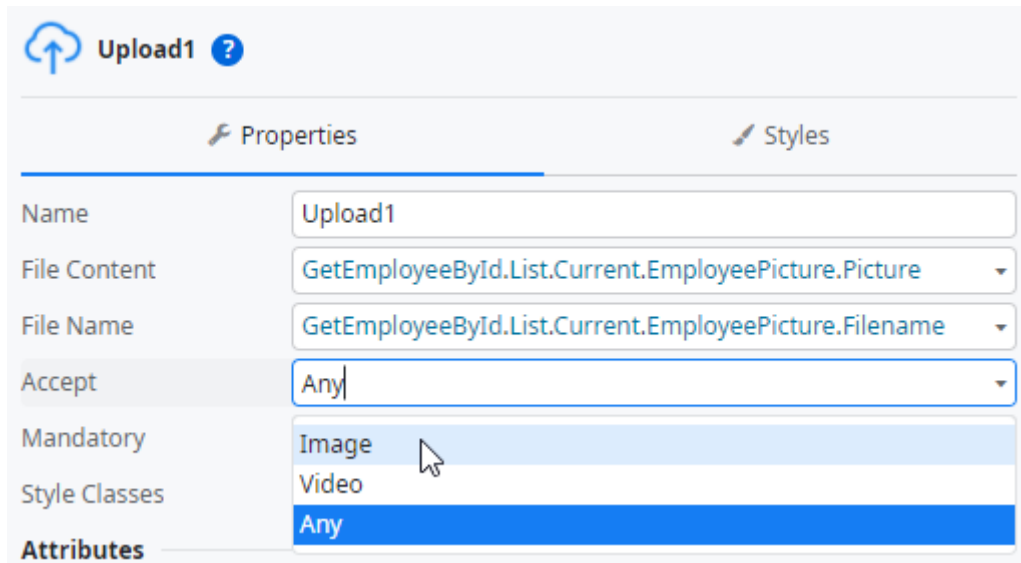
Notice that the **File Content** was automatically connected to the Picture attribute that we dragged from the Aggregate. And now we need to set the **File Name** accordingly.

- Set the **File Name** property to `GetEmployeeById.List.Current.EmployeePicture.Filename`.



Just like it happened automatically with the Picture, we are here setting the File Name property of the Upload element to the Filename attribute fetched from the Aggregate.

- 5) Still on the properties of the Upload element, set the **Accept** property to *Image*.



This will automatically add an Image element attached to the Upload, so we can see a preview of the image on the Screen, as soon as the user uploads the image.




- 6) **Publish** the module to save your latest changes.

- 7) After it is done, click on the blue button that says **Open in browser**, so you can see the changes in the browser. Try to upload some pictures. Don't forget that we're not saving them yet in the database, so the goal is to make sure the picture appears as preview in the EmployeeDetail Screen.

EmployeeDirectory Employees Login

Edit Employee


[Change](#)

Name *
Patricia Wesley

Birth Date *
12/25/1986

Email *
patricia.wesley@example.com

Phone *
1-555-723-3191

Job Position *
Sales Manager West

Bio
Top-ranked sales manager, contributed to record sales and new account develop

Hiring Date *
03/31/2020

Is Active
☒

Department
Accounting

Location
Alain Commercial Park

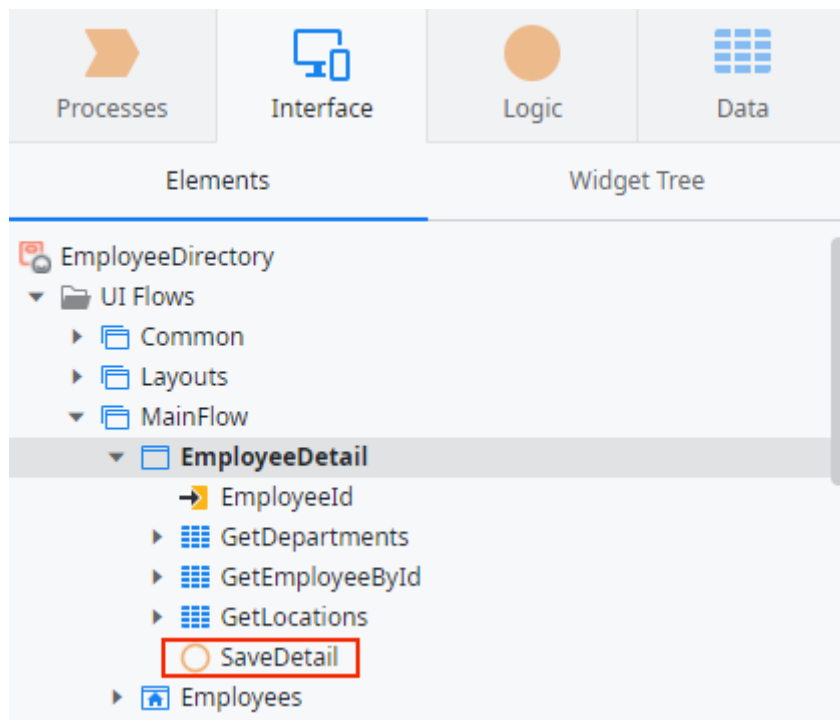
[Back](#) [Save](#)

Saving the Employee Picture to the Database

Now we just need one last part to make this functionality work as expected. You will need to add some logic to save the picture of the employee in the database, whenever the user of the app clicks on the Save button of the EmployeeDetail Screen.

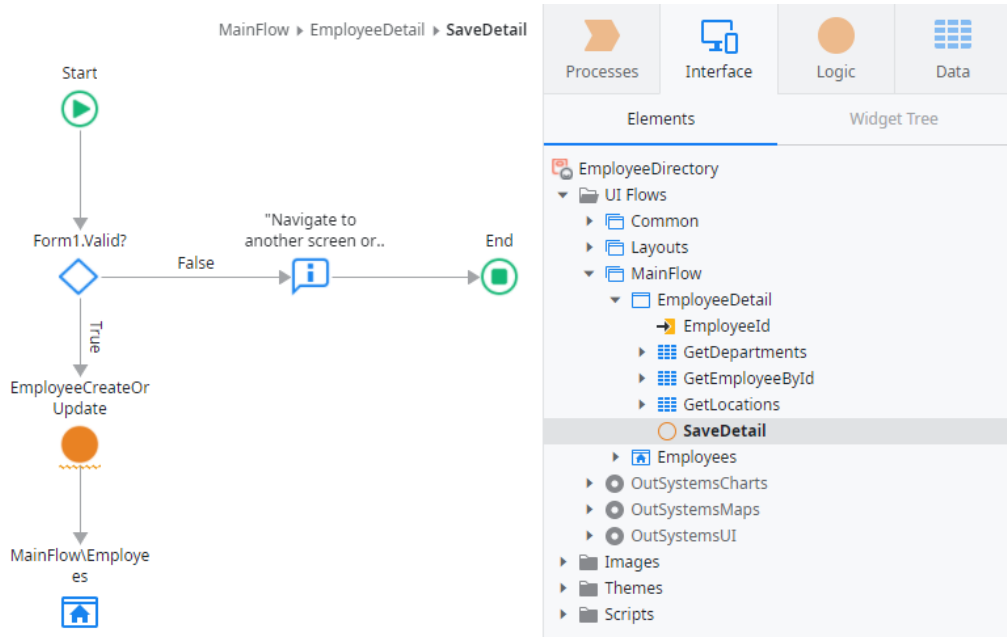
Explore the Logic

- 1) On the right sidebar of Service Studio, expanding the EmployeeDetail Screen, we can find a **SaveDetail** element.



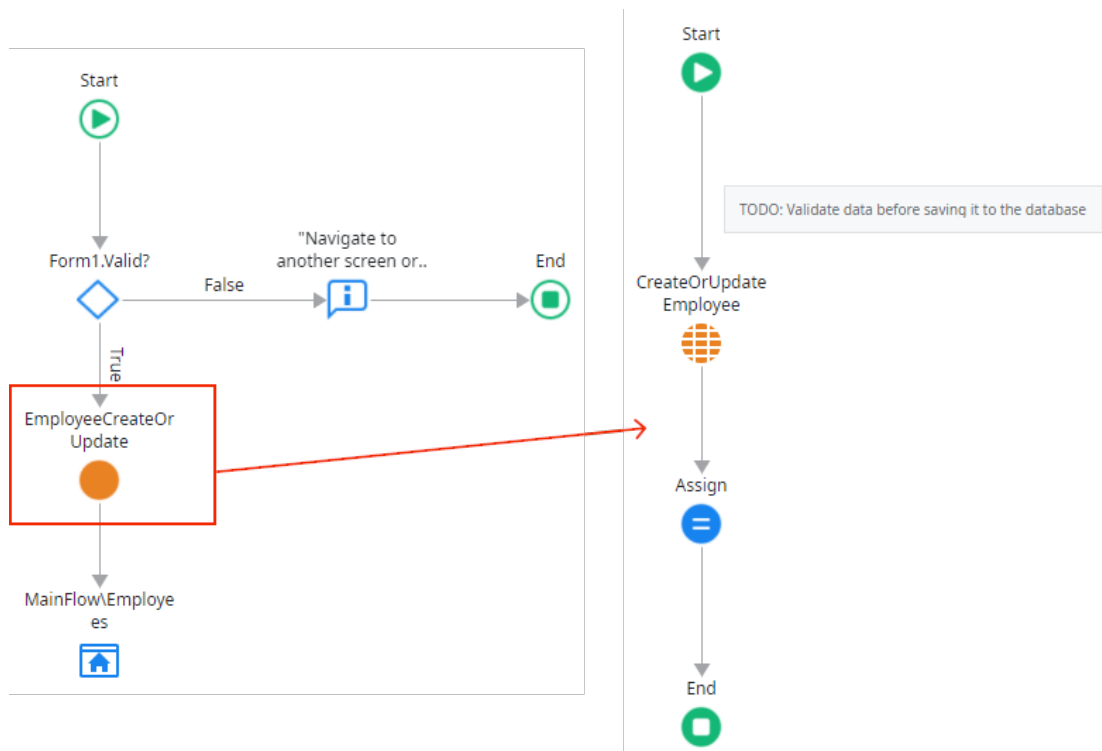
In OutSystems, this element is called **Action**. An Action is where you can create logic flows that are executed in your applications.

2) Double-click the **SaveDetail** Action to open it.

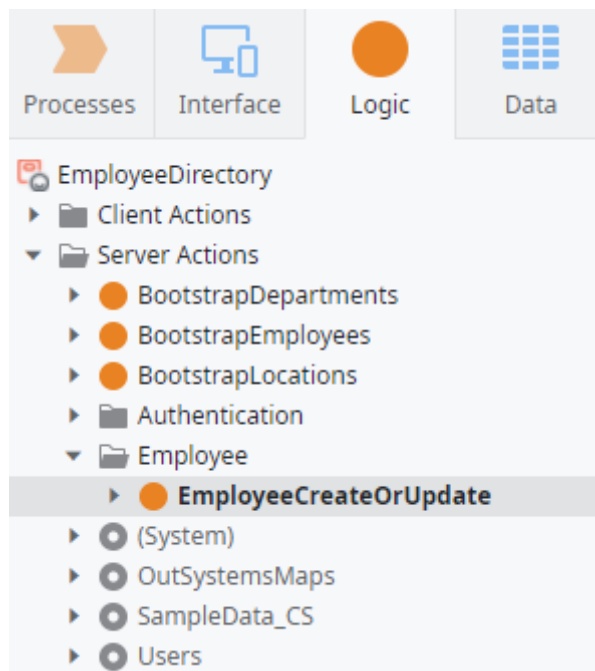


The SaveDetail Action already has some logic built for you. This Action was created automatically when the EmployeeDetail Screen was created and it runs automatically when the Save button on that Screen is clicked. This Action already has some logic created for you, but don't worry about that now. Let's focus on saving the picture!

- 3) Double-click the **EmployeeCreateOrUpdate** element to open it. You should now see a new flow, just like you can see on the right of the next image.

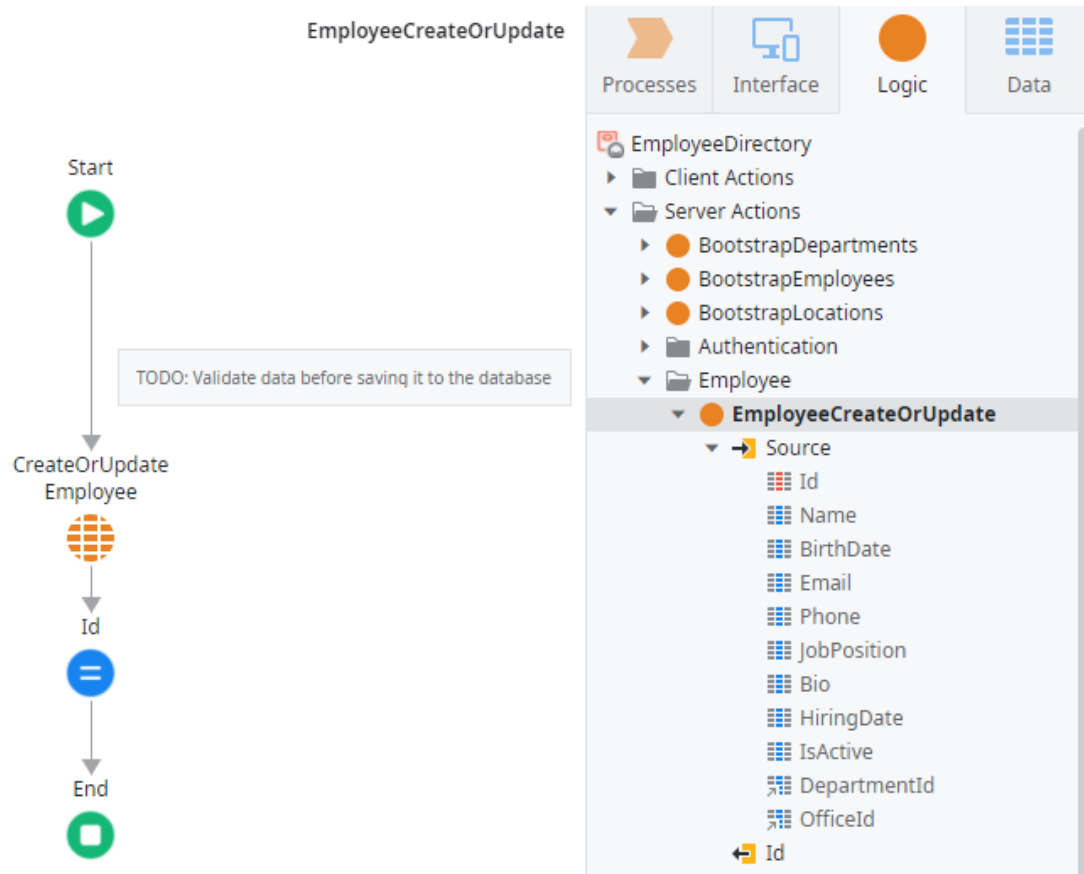


- 4) On the right sidebar of Service Studio, you can see that you're now on the Logic tab, and you can see the **EmployeeCreateOrUpdate** element there as well.



This **EmployeeCreateOrUpdate** element is also an Action and it has the logic to save the employee info in the database already created for you.

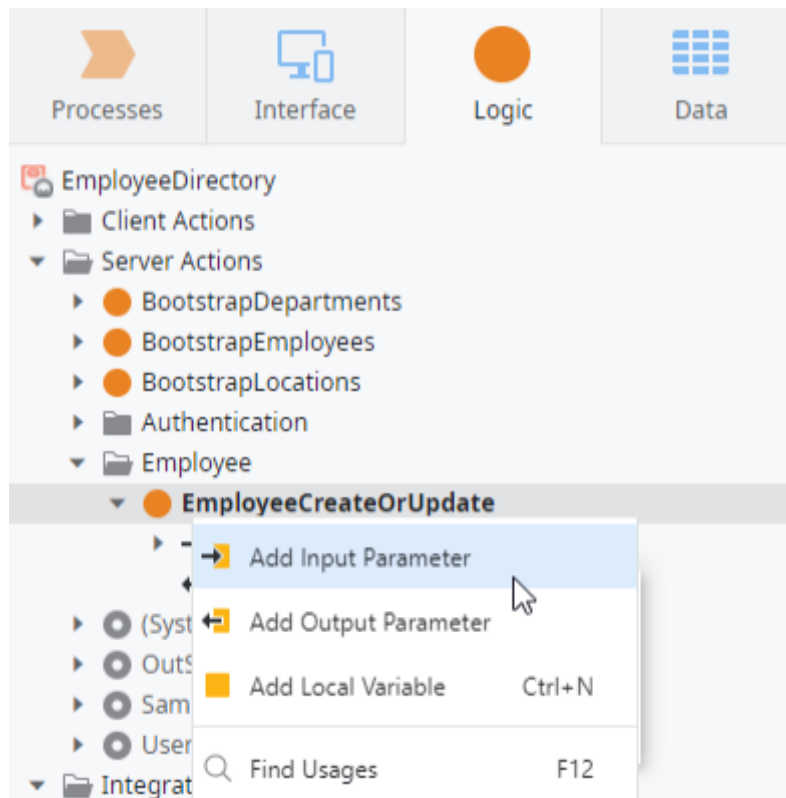
- 5) If we expand the EmployeeCreateOrUpdate element, we can see the input and output parameters for this Action. The Action receives a **Source** as input, which is an Employee record with all the info of one employee. This is the element with the data that the Action will actually save in the database.



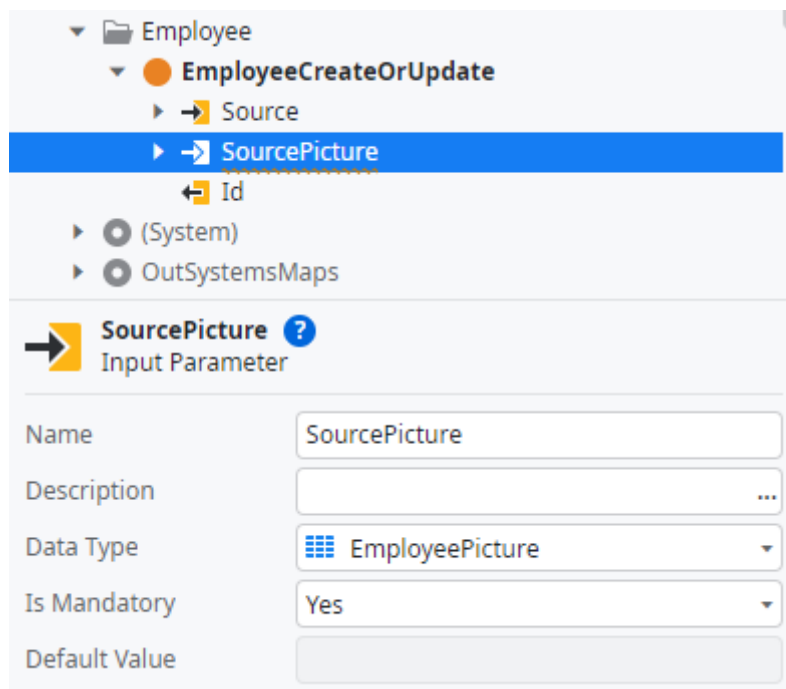
So, now to include the employee picture, you need to extend this logic to also save the picture, starting by adding a new Input Parameter to receive the EmployeePicture information.

Create the Logic to Save the Picture in the Database

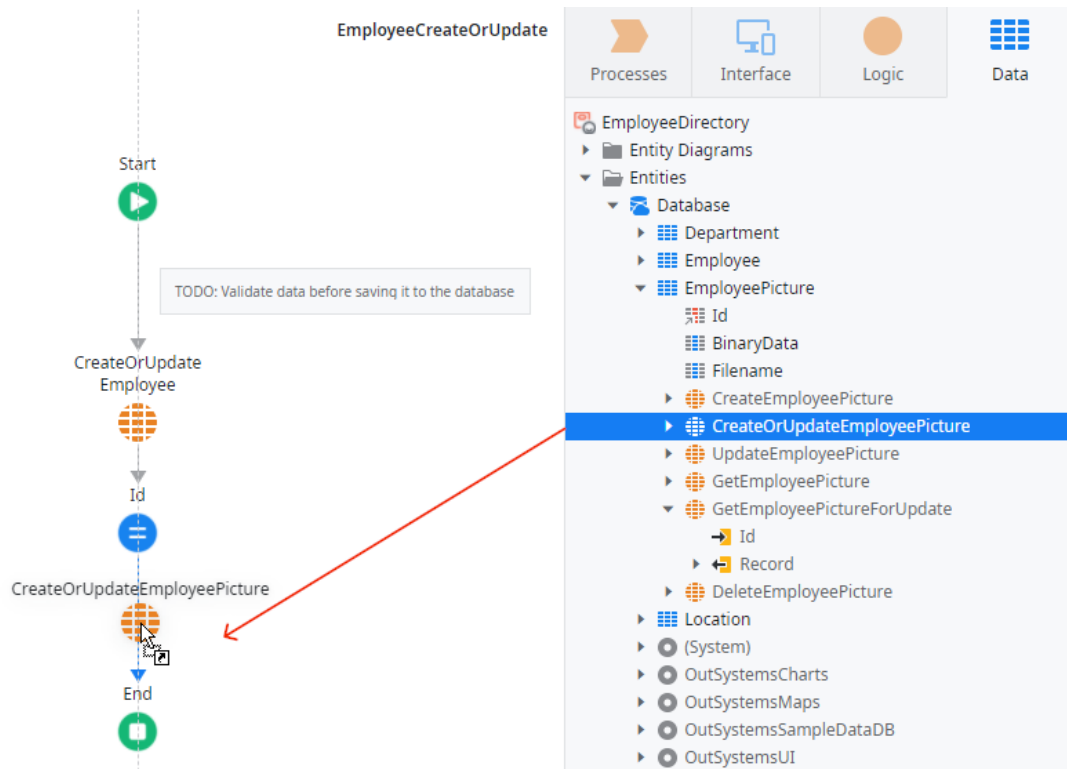
- 1) Right-click the EmployeeCreateOrUpdate Action and select **Add Input Parameter**



- 2) Name it *SourcePicture* and set its Data Type to **EmployeePicture**



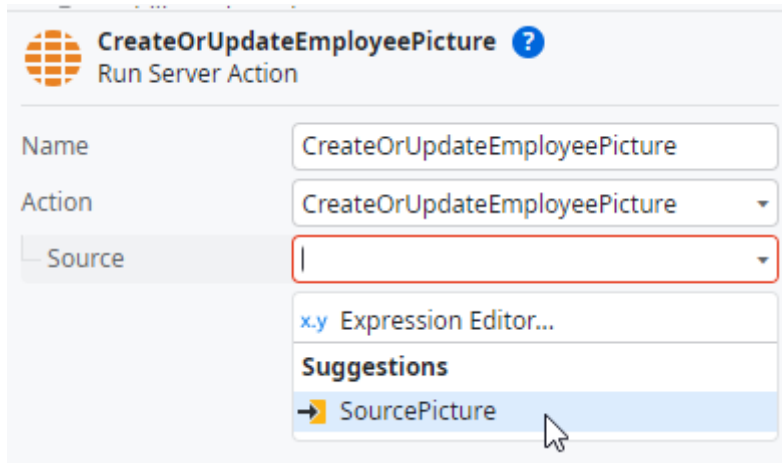
- 3) Switch to the Data tab and expand the EmployeePicture Entity. Then, drag the **CreateOrUpdateEmployeePicture** element and drop it below the Assign and above the End element.



The **CreateOrUpdateEmployeePicture** is an Action that is created automatically when an Entity is created and updates the database with the picture information. We just simply drag it to where we want, and OutSystems does the rest! If you want to know more about these elements, you can check it out [here](#).

- 4) Back in the logic flow that you're working on, click once in the element you just dragged, and look to its properties on the right sidebar. There's an error, since

this Action is expecting a picture to save it in the database. Set the **Source** to be the *SourcePicture* input parameter we created earlier.



CreateOrUpdateEmployeePicture ?
Run Server Action

Name: CreateOrUpdateEmployeePicture

Action: CreateOrUpdateEmployeePicture

Source: [Empty field]

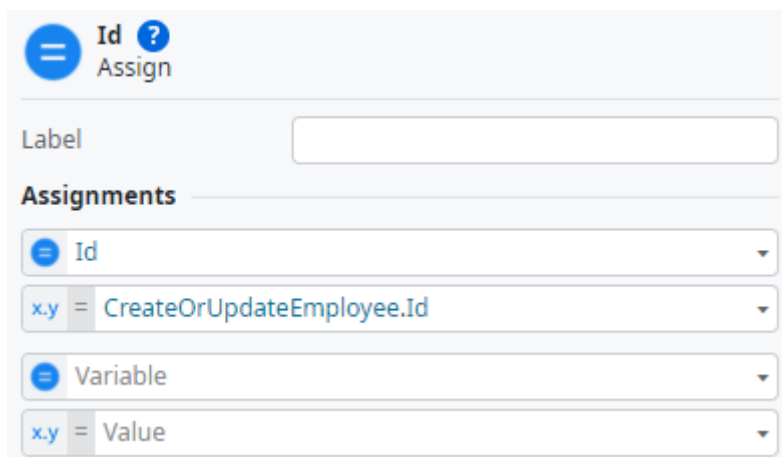
x.y Expression Editor...

Suggestions

- SourcePicture

Here, we are actually sending to the database, the employee picture that the EmployeeCreateOrUpdate Action is receiving via the input parameter you created earlier.

- 5) Now, remember when we said that the Employee and EmployeePicture will share the same identifier? So, now it's the time when we actually make them the same. Click on the blue element on the Action flow. This is an Assign element, which allows having multiple assignments of values to variables.



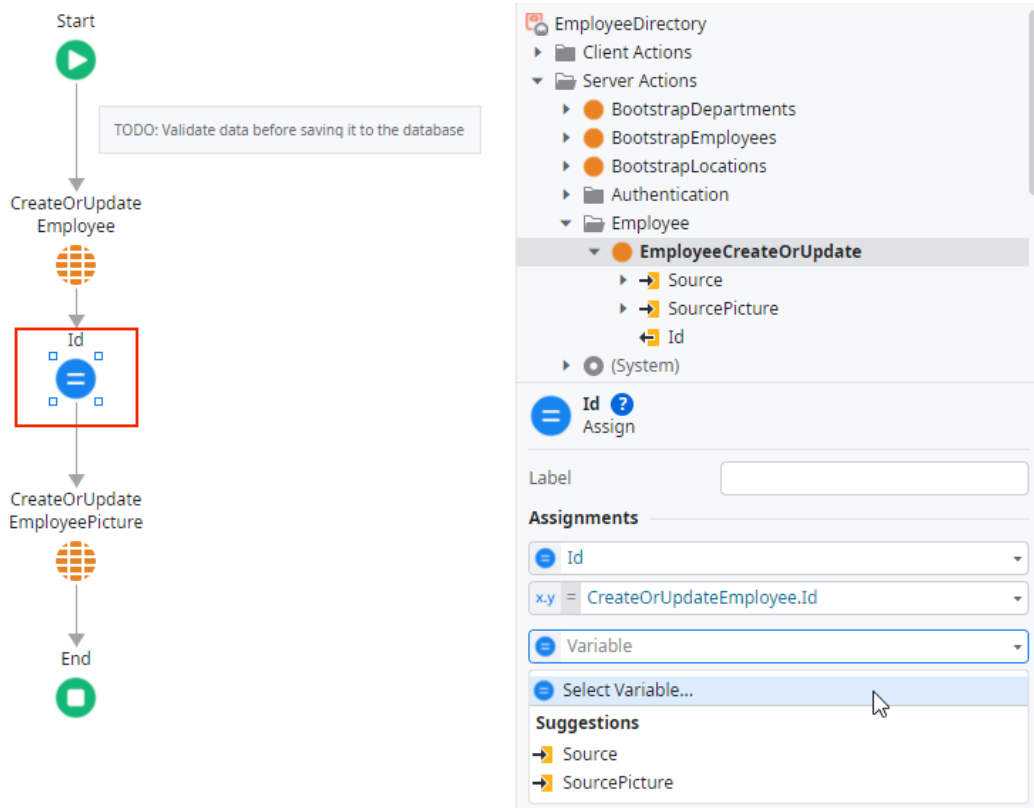
Id ?
Assign

Label: [Empty field]

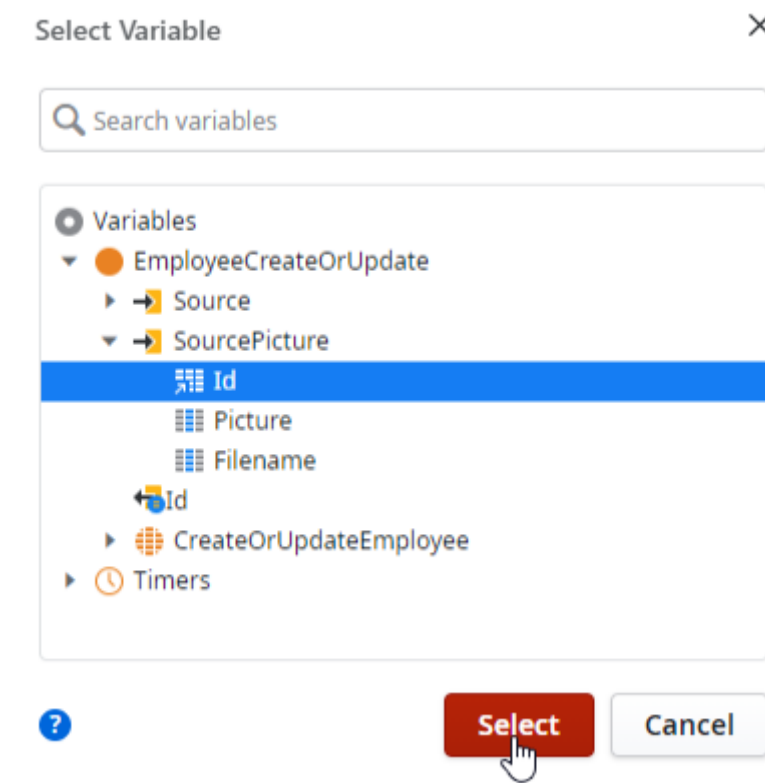
Assignments

- [Blue icon] Id
- x.y = CreateOrUpdateEmployee.Id
- [Blue icon] Variable
- x.y = Value

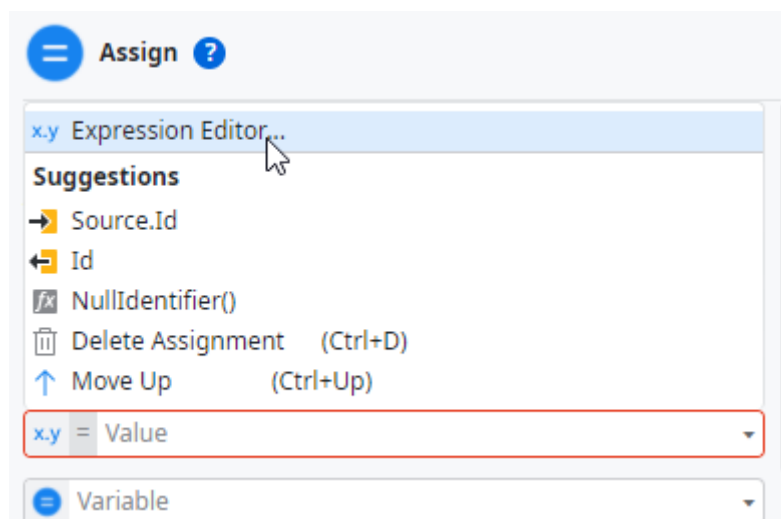
- 6) Within the properties of the Assign element, select the first empty dropdown and click **Select Variable**.



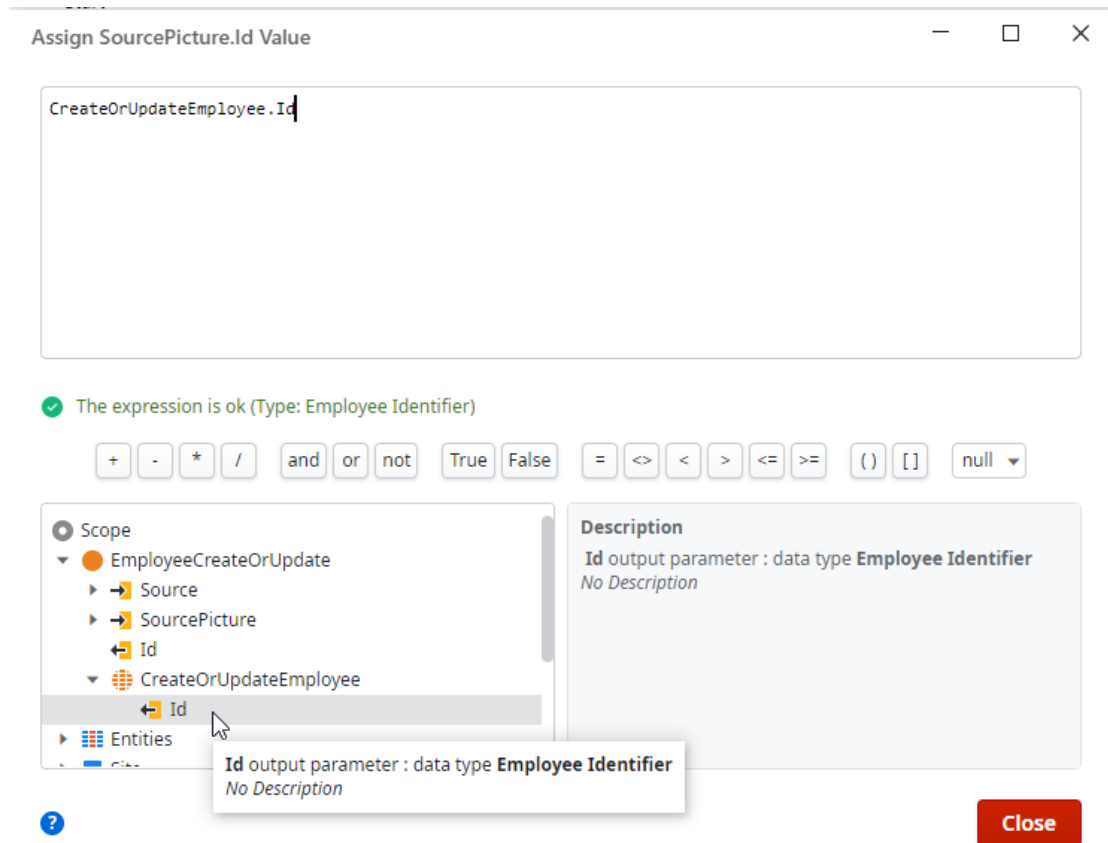
- 7) Choose the **SourcePicture Id** and click Select.



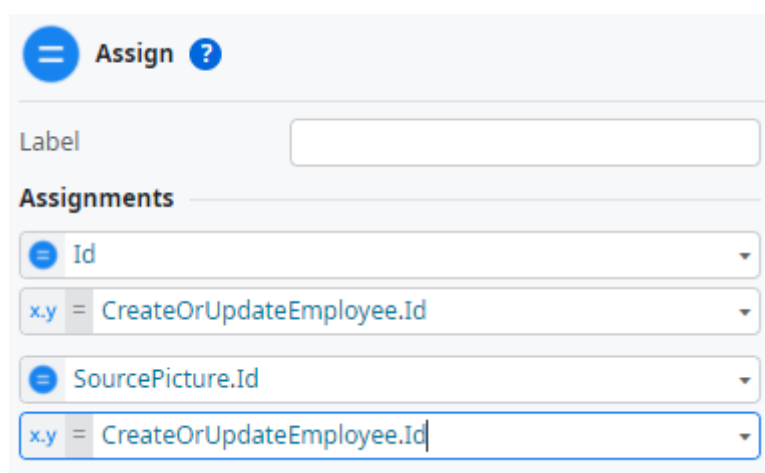
- 8) In the dropdown below (value field), select **Expression Editor**.



- 9) In the new dialog, expand the **CreateOrUpdateEmployee** Action under Scope, and select the **Id** by double-clicking on it. Make sure the expression is fully visible in the text box above. When it's there, click Close.



- 10) The Assign element should look like this:

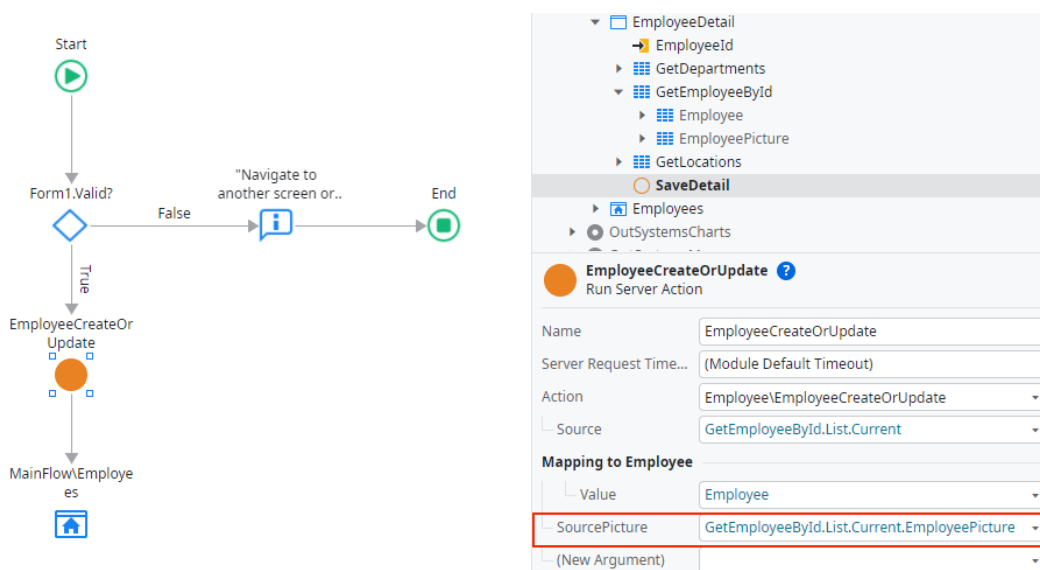


Finish the Logic and Test the App

There's only one step left to finish. This final step will tie everything together and

guarantee that the picture uploaded by the user, in the EmployeeDetail Screen, is actually being sent to the database.

- 1) Go back to the **SaveDetail** Action, in the Interface tab and under the EmployeeDetail Screen, and double-click on it to open. Click once on the **EmployeeCreateOrUpdate** element in the flow. Notice that we now have an error, since the Action is expecting a SourcePicture, which is exactly the Input Parameter we added before for the picture. Set the **SourcePicture** property as *GetEmployeeById.List.Current.EmployeePicture*.



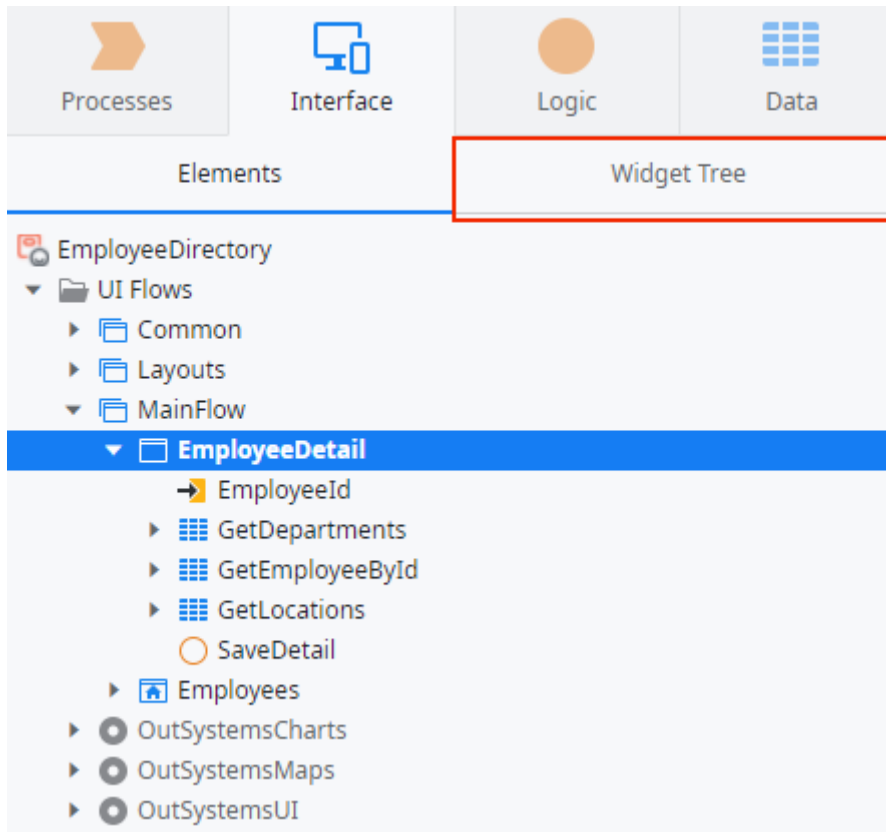
- 2) Publish the module and test in on the browser. Now you should be able to add a Picture to an Employee. After that, click on the Save button in the Detail Screen, and go back to the Employee List, then back again to the same Employee Detail page that you have just modified. Is the picture still there? If yes, you have successfully saved the Picture. Congrats!!!

Customizing the Layout

At this point, the new functionality works, but the layout could still be improved. So the next steps will show you how to customize it. We will remove the "empty" column on the right and redistribute the Form's fields, making sure the Form occupies the full width of the Screen. You can be as creative as you want! This is just an example!

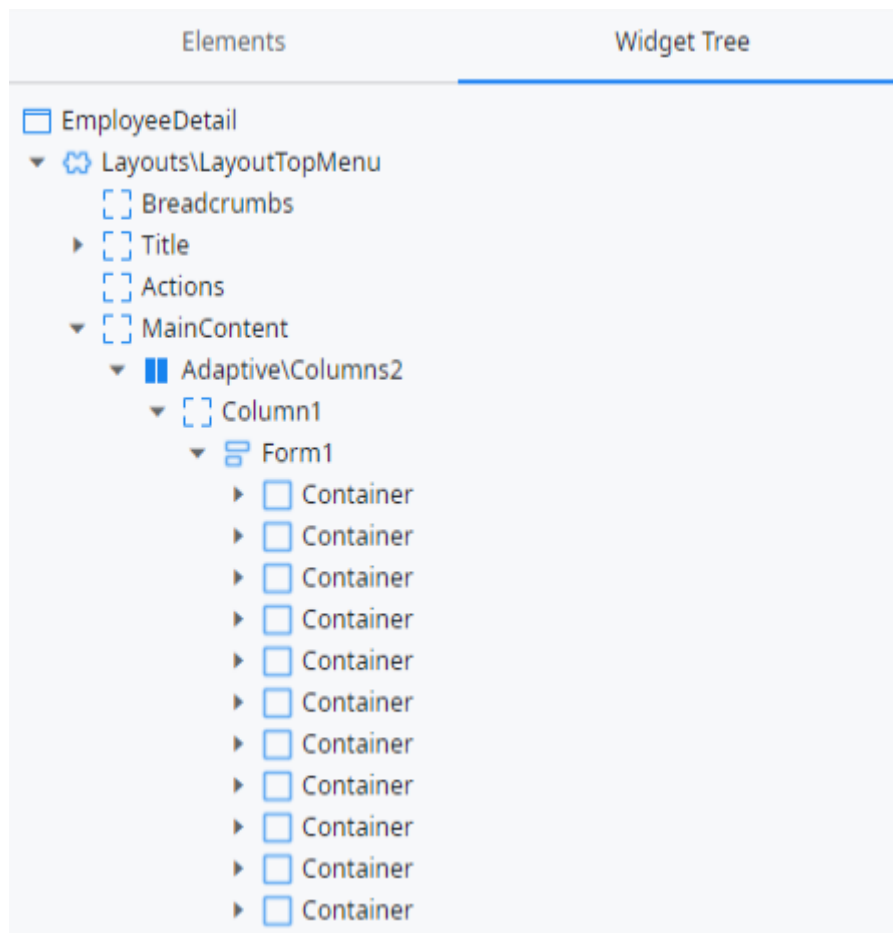
- 1) Go back to Service Studio, make sure you are in the Interface tab, and double-click the EmployeeDetail Screen to open it.

- 2) Click on the **Widget Tree** option to see the Screen structure.



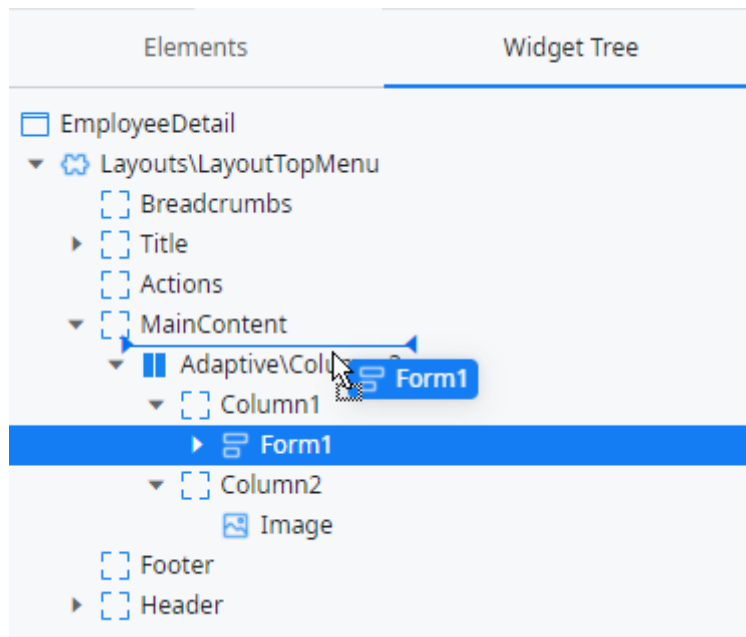
This option appears every time we have a Screen opened on Service Studio. It shows the structure of all the elements inside the Screen, and it is a great helper to set the pieces right where they belong.

- 3) Expand the MainContent area, then every first element nested inside it, until you get to the **Containers** inside the Form. This is the page structure so far.



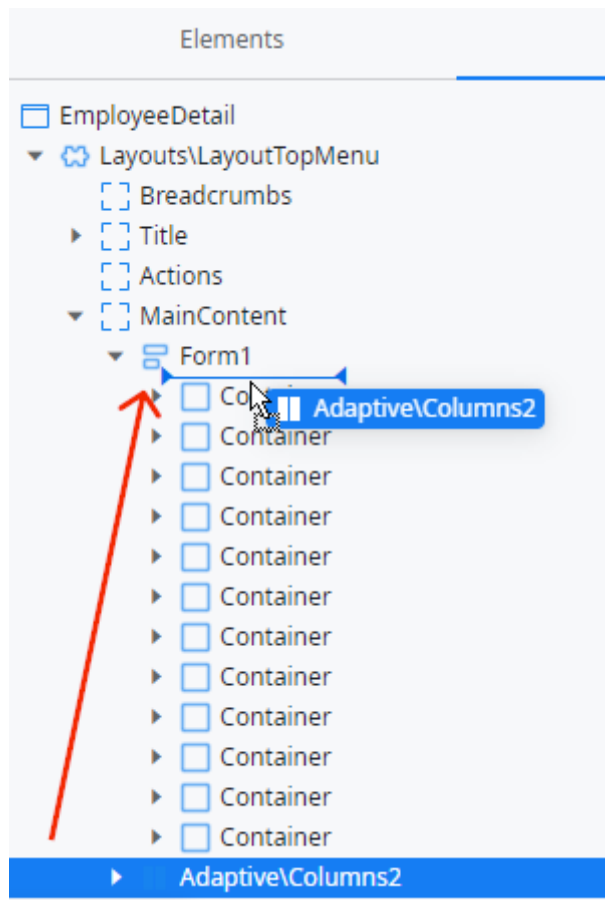
- 4) Select the **Form1** by clicking on it, and dragging it **above the Adaptive\Columns2** element, just like the following image demonstrates. This

Adaptive\Columns2 element is helping us divide the Screen into 2 columns. This change will make the Form fill the entire width of the page.

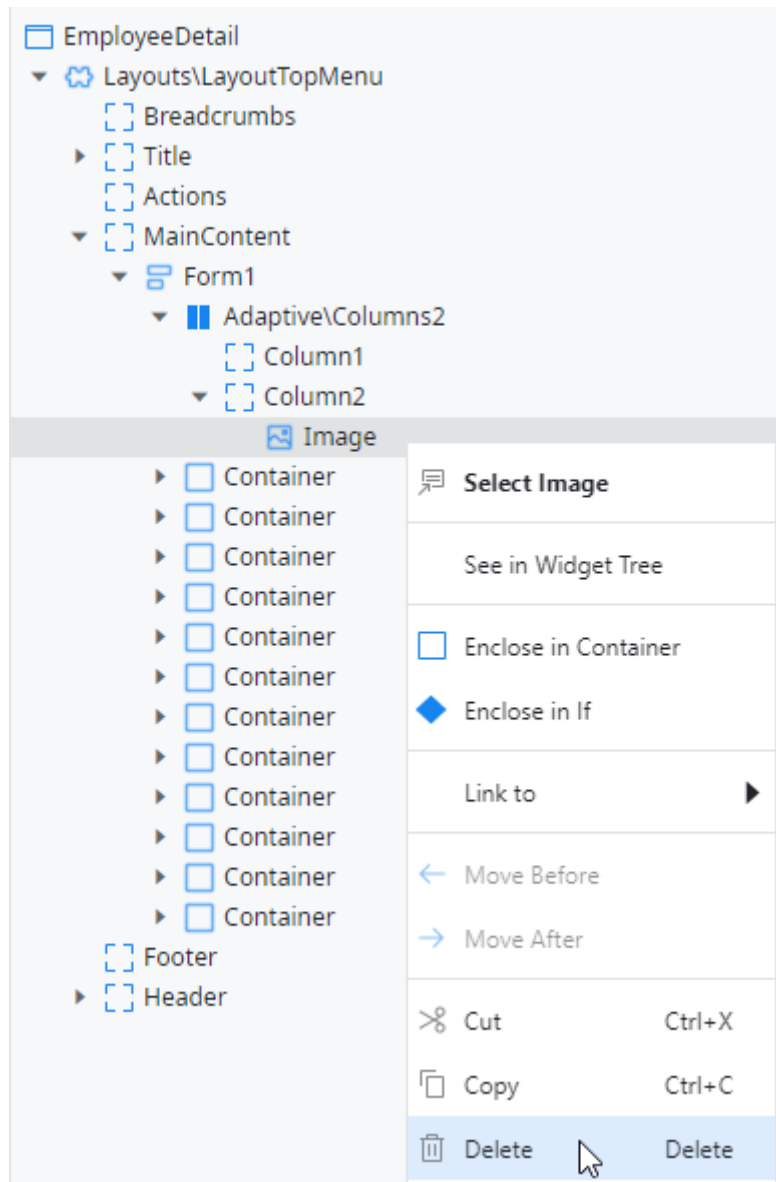


- 5) However, we don't want you to get rid of the Columns2 element. We want to make sure the Form itself has a separation into 2 columns, where some fields

will be on the left and some on the right. So, let's move the Columns2 element **inside Form1**, and make it the first element inside it.

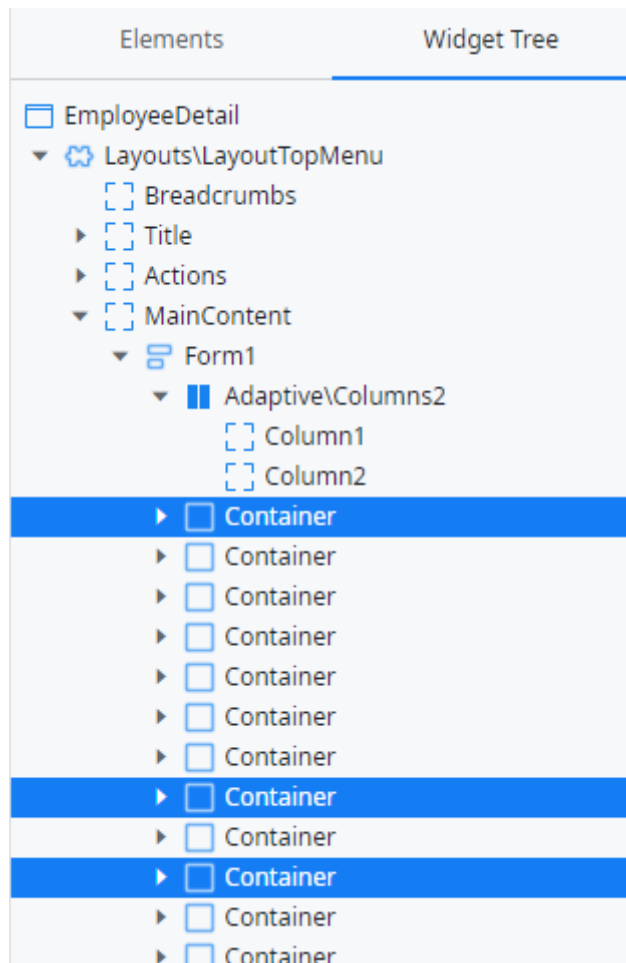


- 6) Expand the Adaptive\Columns2 element, and the **Column2** inside it, and delete the Image. This is the **Image** that appeared on the right column of the page and we don't want it.

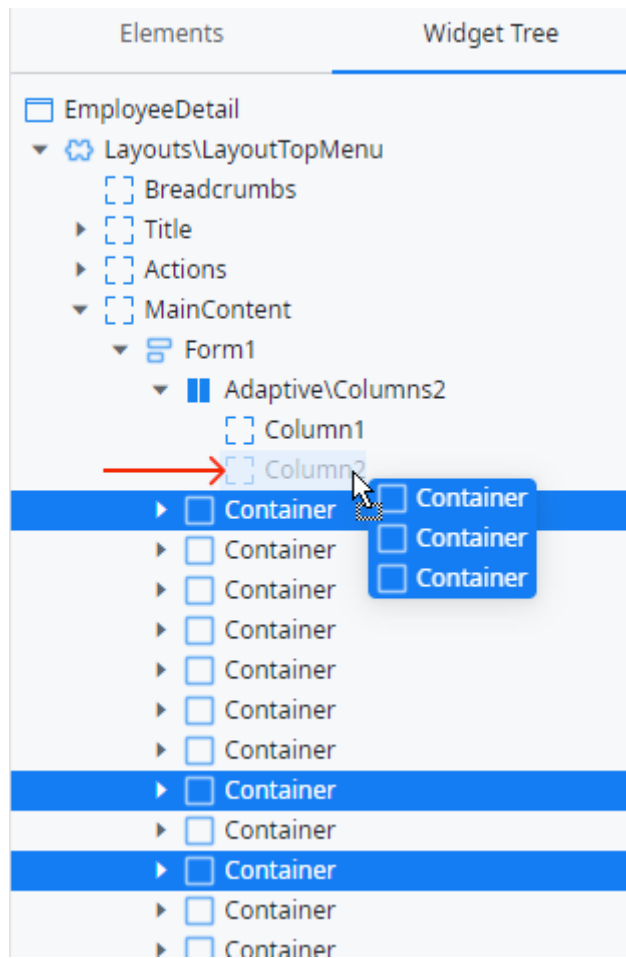


- 7) Now, let's distribute the fields. Each Container is a field in the Form, with a Label and an input element inside it. Since we have a lot of Containers, let's use some numbers. Select the first, seventh and ninth Containers, by using CTRL+click.

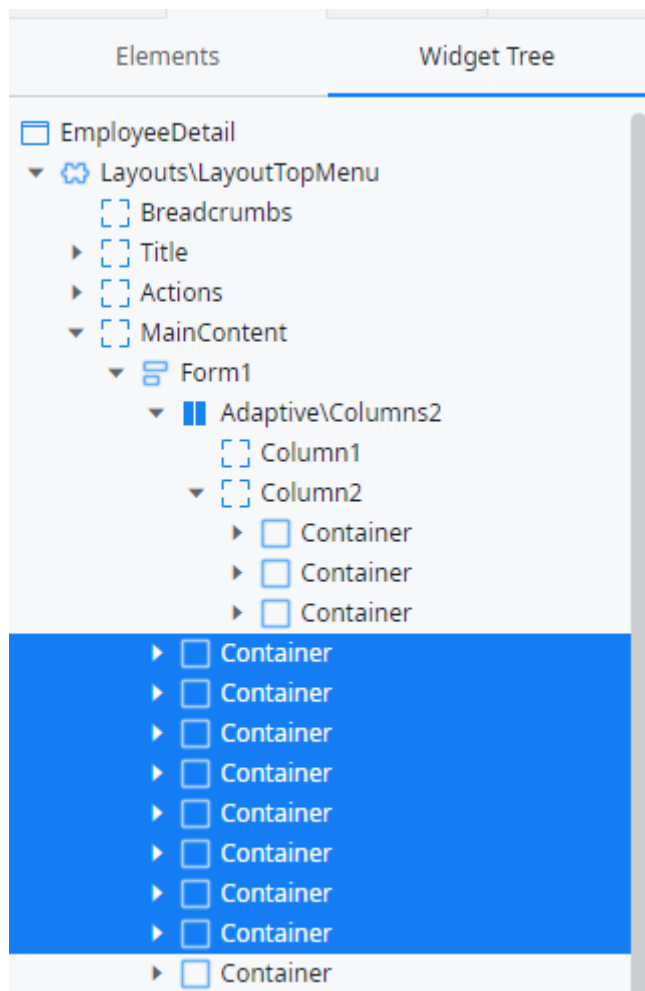
Let's search for the Container representing the Picture (first), Bio (seventh) and IsActive (ninth) fields.



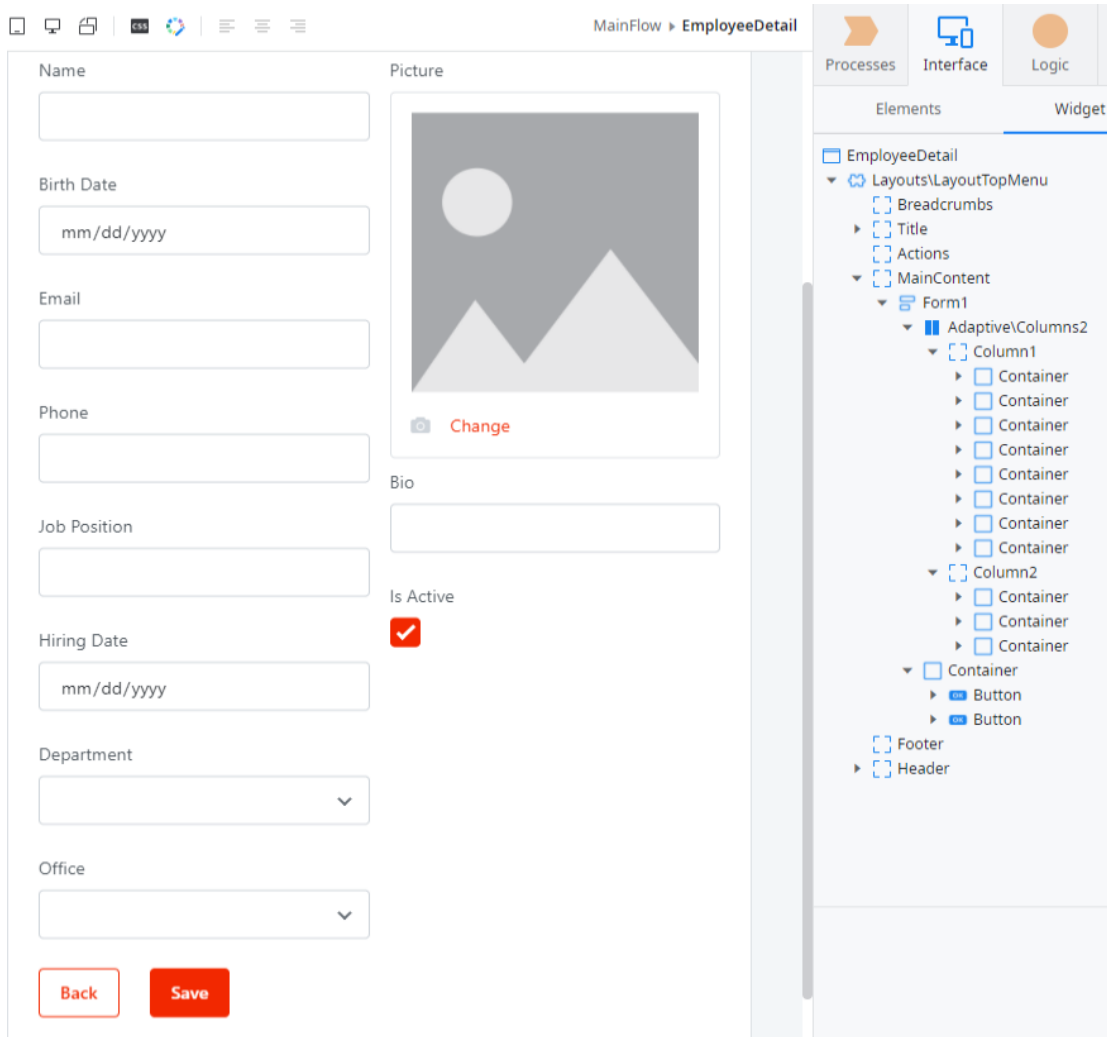
- 8) Now, drag the selected Containers **inside the Column 2** section.



- 9) Select the remaining Containers, except for the last Container. This one has the Save and Back buttons and will stay outside of the columns.



10) Drag them and drop them **inside Column1**. You should get something like this:



11) Now publish and test your app in the browser! And feel free to adapt and customize the layout if you want to.

Wrapping Up

Congratulations on finishing this tutorial. With this exercise, we had the chance to go through some essential aspects of OutSystems and get to know more about the platform, in particular the Logic side.

References

If you want to deep dive into the subjects that we have seen in this exercise, we have prepared some useful links for you:

- 1) [Aggregates 101](#)
- 2) [Modeling Data Relationships](#)
- 3) [Reactive UI Development 101](#)
- 4) [Logic](#)
- 5) [Upload Widget](#)

See you on the next tutorial!