

Fetching Data On Demand Exercise

Table of Contents

Outline.....	2
Resources	2
Scenario	3
How-To.....	6
Getting Started	6
Change the Data Model	10
Add the New Fields to the EmployeeDetail Screen	11
Fetch States and Cities on Demand	15

Outline

In this exercise, we will extend an existing application to fetch data only on demand. The application lists all the employees of a company and allows registered users to access and modify the personal data.

In this exercise, we will:

- Enable the application to store the country, state, and city where the employee is located.
- Allow registered users to assign a country, state, and city to an employee.
- Ensure that when a country is selected, only the respective states appear.
- Ensure that when a state is selected, only the respective cities appear.

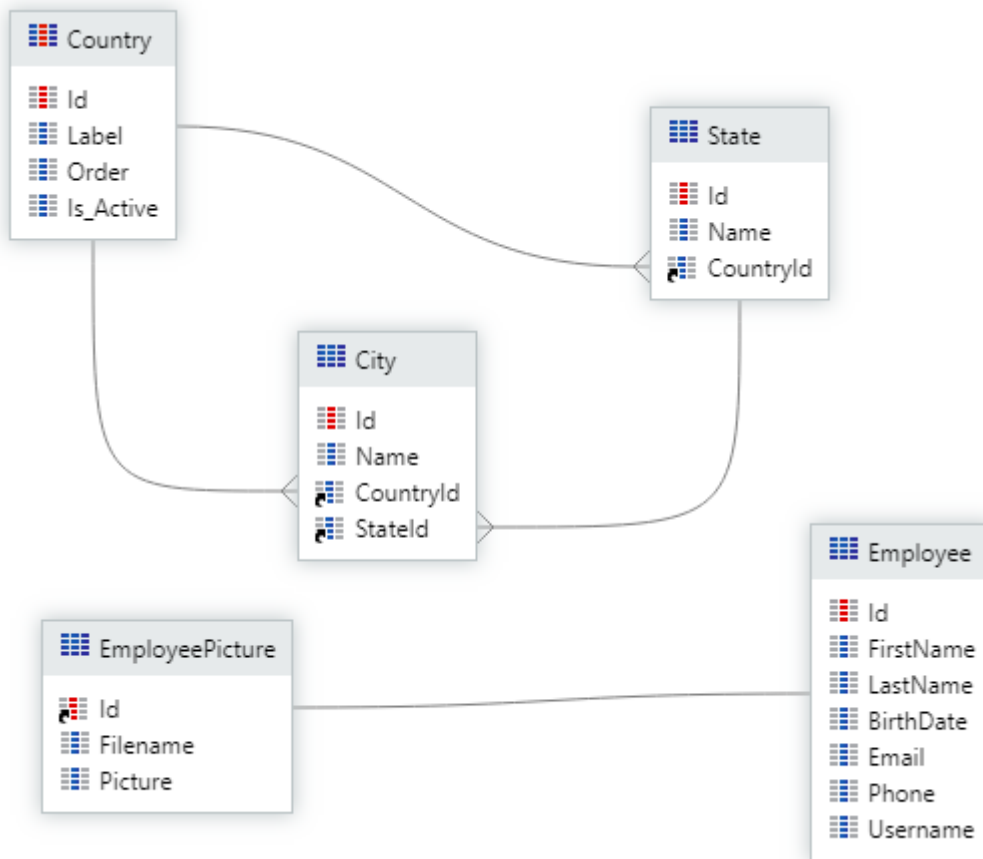
Upon completion, we'll have created a few aggregates: some will be triggered at start, some only on demand, and some logic to explicitly trigger the right aggregates at the right time.

Resources

This exercise has a Quickstart application already created, and the application has everything you need to start the exercise. This Quickstart application can be found in the Resources folder of this exercise, with the name **Fetching Data On Demand Exercise.oap**.

Scenario



In this exercise, we'll start from an existing application with one module. Inside that module are five Entities already populated with data.



The application has two screens: Employees and EmployeeDetail. The first one lists the employees in the database, while the second displays the details about an employee, including a picture. Both screens are only accessible by **Registered** users. In the

EmployeeDetail Screen, it is also possible to edit the information about the employee, which is already implemented in the **SaveOnClick** Action.


Lawrence Ricci

First Name *	<input type="text" value="Lawrence"/>	
Last Name	<input type="text" value="Ricci"/>	
Email	<input type="text" value="LawrenceCRicci@dayrep.com"/>	
Phone	<input type="text" value="818-329-1436"/>	
Birth Date	<input type="text" value="mm/dd/yyyy"/>	Picture
		 <input type="text" value="Stlef1958.png"/>
<input type="button" value="Save"/>		

In this exercise, we will extend this application with some additional information about the employee, related to location. So, we have the Country, State, and City Entities already created with data that we can leverage for our purposes.

First, we need to associate a country, state, and city with each employee and allow registered end-users to select a country, state, and city for each employee. The EmployeeDetail screen should look something like this:

Lawrence Ricci

First Name *	<input type="text" value="Lawrence"/>	
Last Name	<input type="text" value="Ricci"/>	
Email	<input type="text" value="LawrenceCRicci@dayrep.com"/>	
Phone	<input type="text" value="818-329-1436"/>	Picture
		<input type="text" value="Stlef1958.png"/>
Birth Date	<input type="text" value="mm/dd/yyyy"/>	Country
		<input type="text" value="United States Of America"/>
		State
		<input type="text" value="Alabama"/>
		City
		<input type="text" value="Birmingham"/>

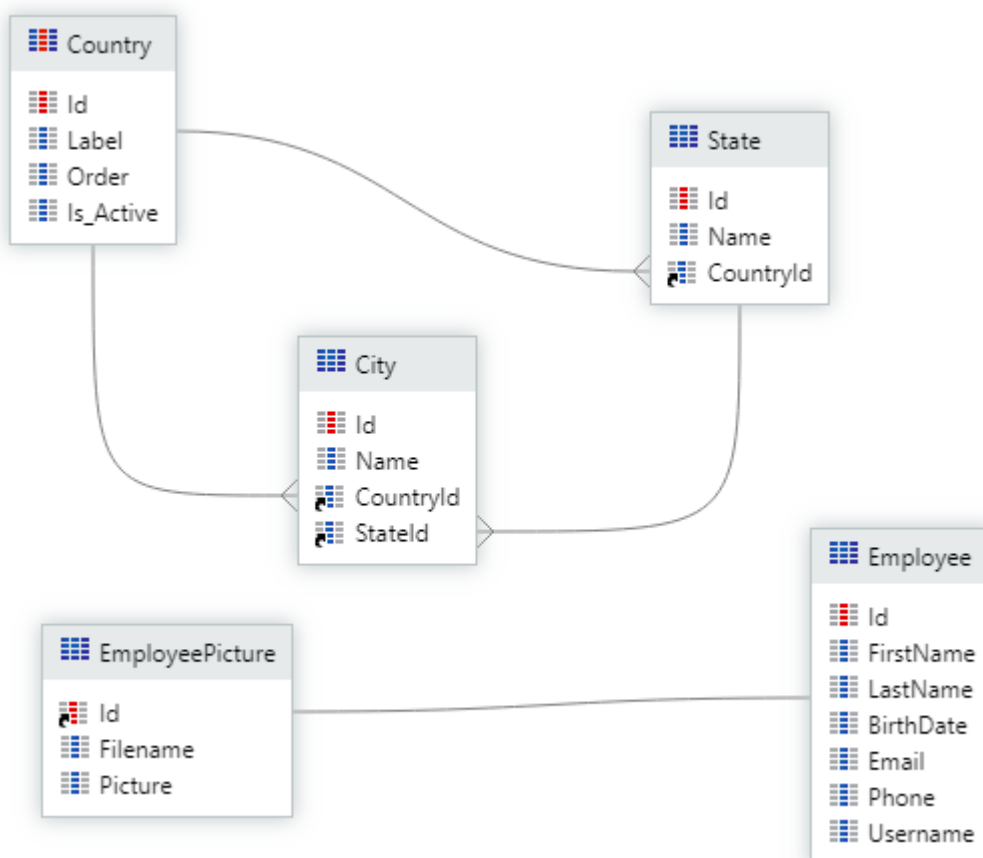
To implement this behavior, we need to fetch all the countries, states, and cities from the database. However, we need to be careful. One of the requirements of the application is to make sure that only the states of a particular country and cities of a particular state appear in the respective dropdowns. For example, if the end-user chooses United States of America, only the US states should appear in the dropdown. Then, if the user chooses California, only cities in that state should appear in the third dropdown.

How-To

In this section, we'll show you how to do this exercise, with thorough, step-by-step instructions. **If you already finished the exercise on your own, great! You don't need to do it again.** If you didn't finish the exercise, that's fine! We are here to help you.

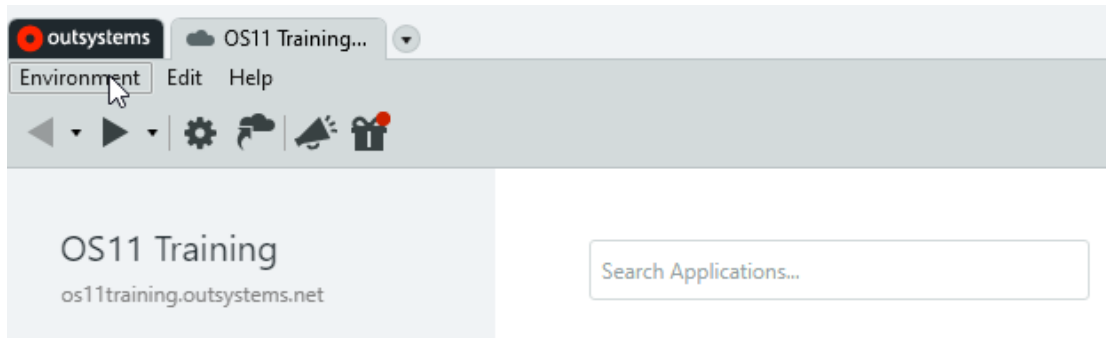
Getting Started

To start this exercise, we need to install the Quickstart file, which already has the created application with two Screens and the defined data model.

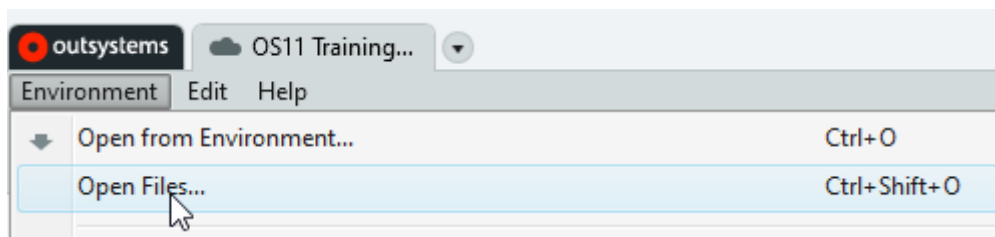


Before we start, we need to install the Quickstart application in our development environment. You'll need to have Service Studio open and connected to an OutSystems Environment (e.g. Personal Environment).

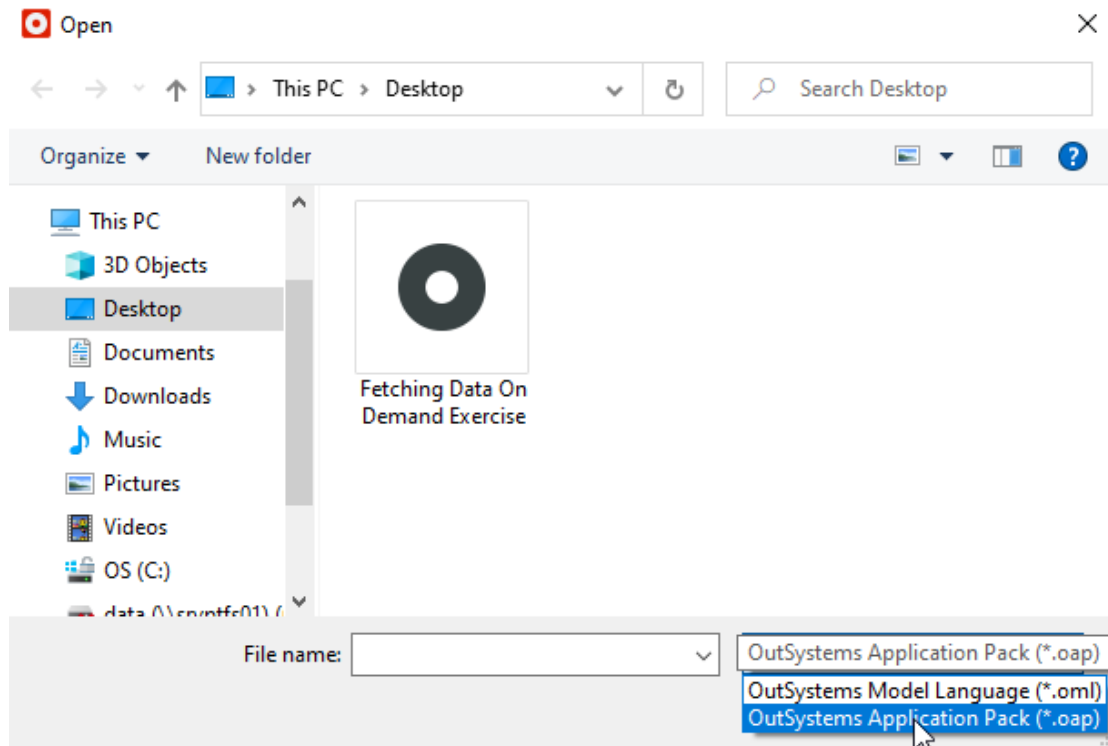
- 1) In Service Studio's main window, select the **Environment** menu on the top left.



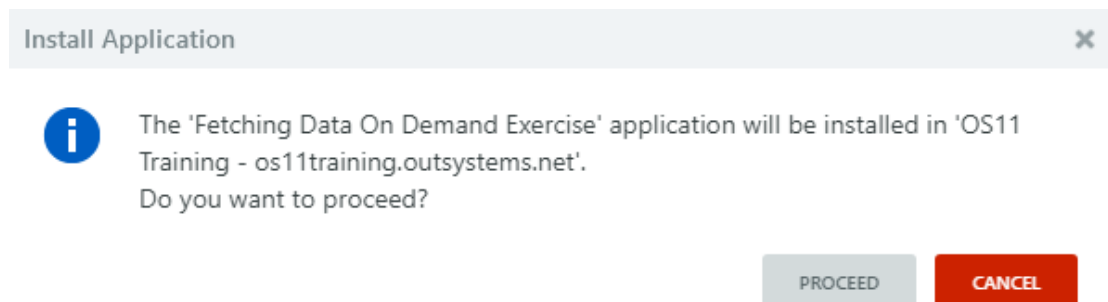
- 2) Select **Open Files...**



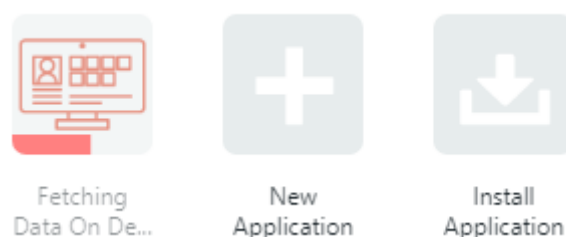
- 3) In the following dialog, change the file type to OutSystems Application Pack (.oap), find the location of the Quickstart, and open the file named **Fetching Data on Demand Exercise.oap**



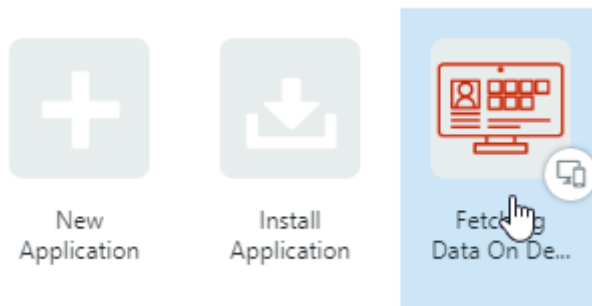
- 4) In the new confirmation dialog, select **Proceed**



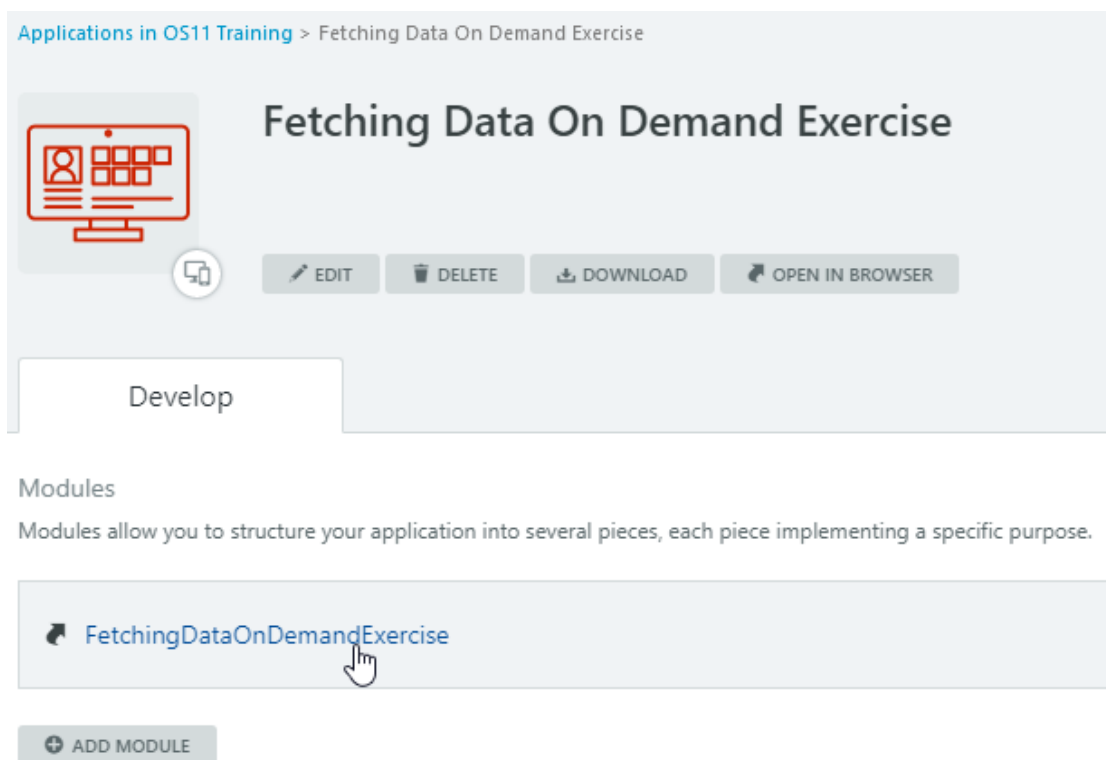
- 5) The application will begin installing automatically. When it's finished, we're ready to start!



6) Open the application in Service Studio.



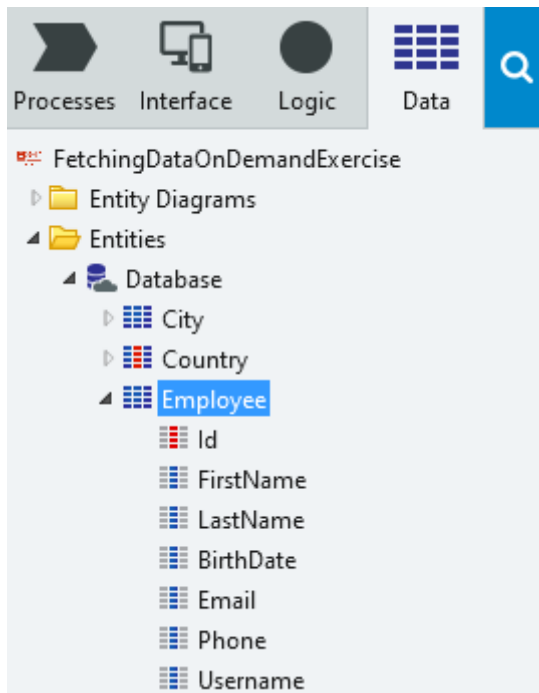
7) The application has only one module. Let's open it!



Change the Data Model

In this section, we'll start by adapting the data model to associate the country, state, and city to the employee.

- 1) Open the **Data** tab and expand the Employee Entity.



- 2) Right-click the Employee Entity and select **Add Entity Attribute**. Set the **Name** of the attribute to *CountryId* and make sure its data type is set to *Country Identifier*. The attribute should be **non-mandatory**.

CountryId Entity Attribute	
Name	CountryId
Description	...
Label	Country
Data Type	Country Identifier ▼
Is Mandatory	No ▼
Delete Rule	Protect ▼

- 3) Add two new attributes: *StateId* and *CityId*. Make sure the data types are set to *State Identifier* and *City Identifier*. Both attributes should be **non-mandatory**.

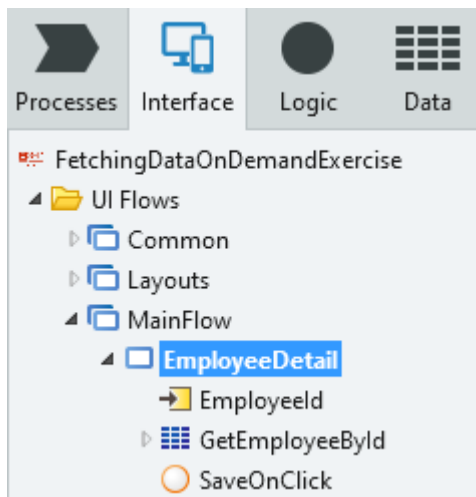
- 4) Publish the module to update the Entity in the database with the three new attributes.



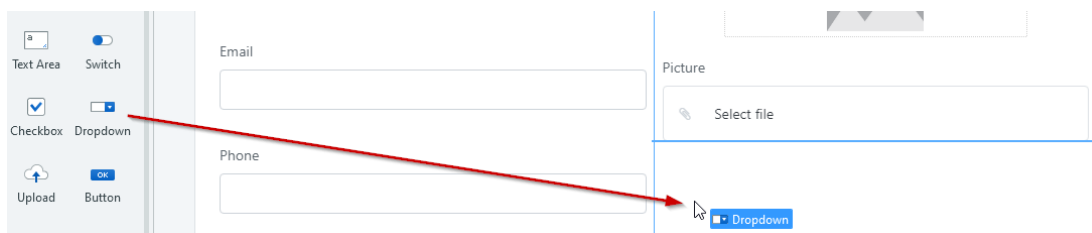
Add the New Fields to the EmployeeDetail Screen

In this section, we will add three Dropdowns to the EmployeeDetail screen: one to select the country, one for the state, and one for the city. To make this work, we need to add a couple of Aggregates to the screen and adjust the Dropdown properties accordingly.

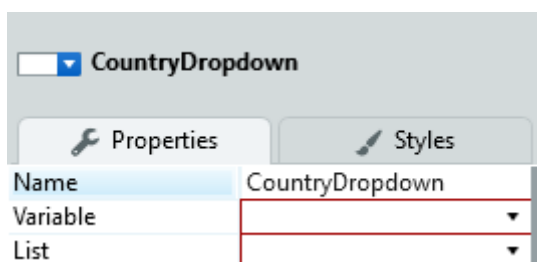
- 1) Switch to the Interface tab and open the **EmployeeDetail** screen.



- 2) Drag a **Dropdown** below the Picture upload field.



- 3) Set the Dropdown **Name** to *CountryDropdown*.

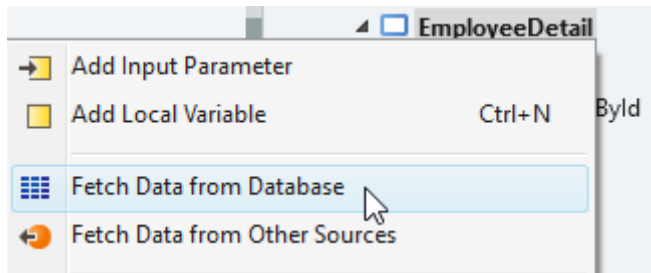


- 4) Set the **Variable** property of the Dropdown to

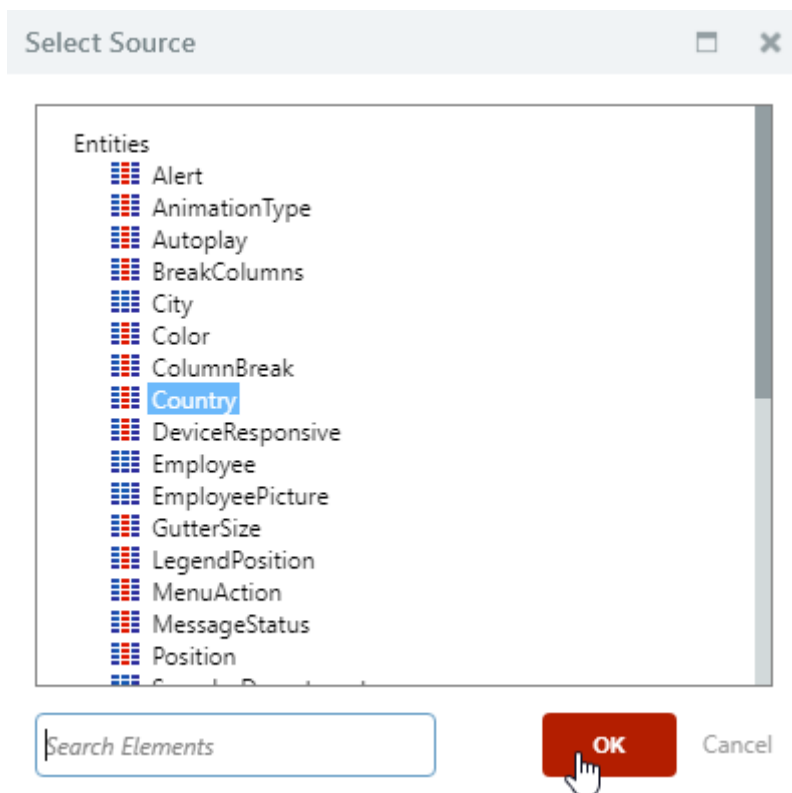
```
GetEmployeeById.List.Current.Employee.CountryId
```

This associates the employee with the country selected.

- 5) Right-click on the EmployeeDetail screen and select **Fetch Data from Database**.



- 6) Click on the canvas and select the **Country** Entity as source of the Aggregate.



- 7) Set the **List** property of the CountryDropdown to:

```
GetCountries.List
```

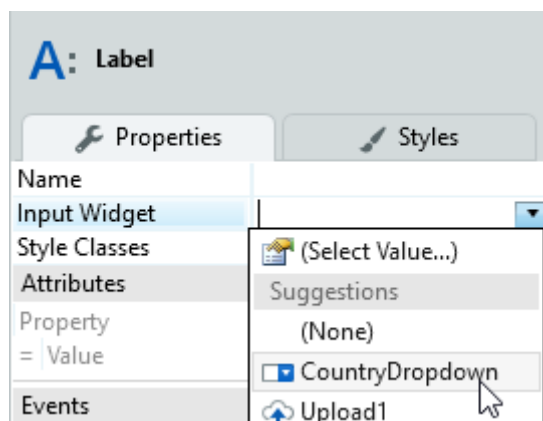
Confirm that you have filled the remaining attributes with the proper information.

CountryDropdown	
Properties	
Name	CountryDropdown
Variable	GetEmployeeById.List.Curren
List	GetCountries.List
Options Content	Text Only
Options Text	Country.Label
Options Value	Country.Id
Mandatory	False
Enabled	True
Empty Text	
Style Classes	"dropdown"

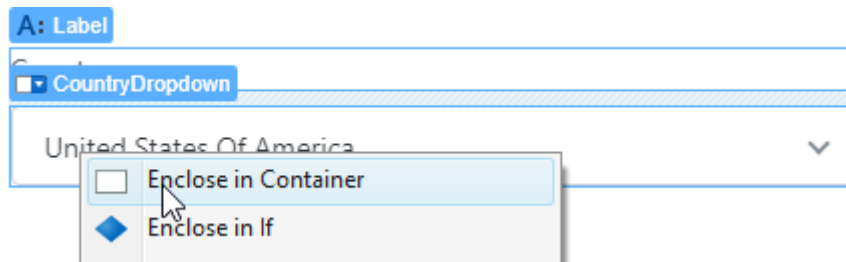
- 8) Set the **Empty Text** property of the dropdown to "-"
- 9) Drag a **Label** Widget right above the CountryDropdown.



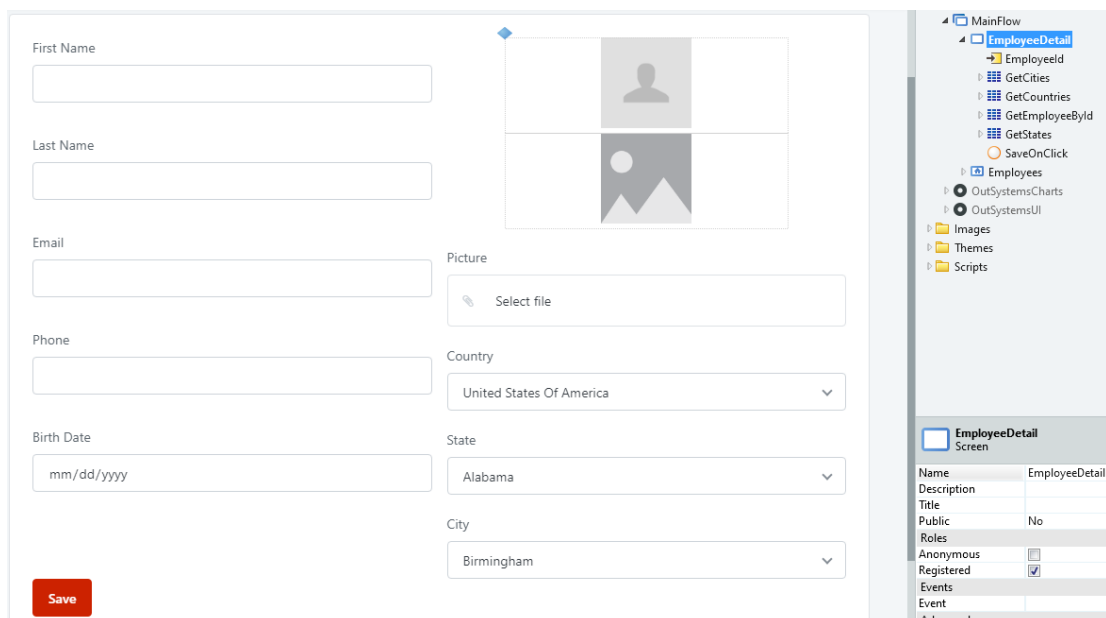
- 10) Set the Label text to *Country*.
- 11) Associate the Label to the CountryDropdown, by setting its **Input Widget** property appropriately.



- 12) Right-click on both the Label and the CountryDropdown and select **Enclose in Container**.



- 13) Repeat the previous steps for the states and the cities. For each field, create a **Dropdown** with a **Label** associated to it and an **Aggregate** to fetch the necessary data to display in the Dropdown. The previewer in Service Studio should look something like this:



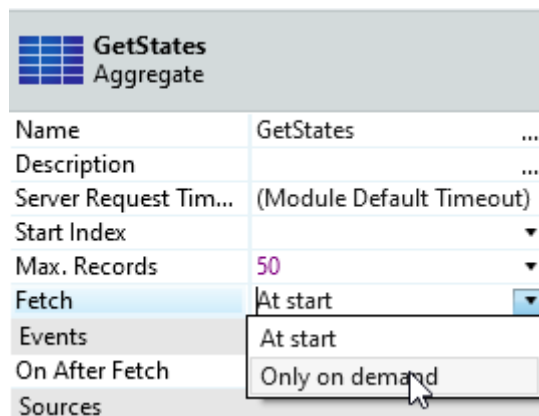
- 14) Publish the module and open the application in the browser.
- 15) Select an employee to open a specific EmployeeDetail Screen.
- 16) Select *Australia* as the Country.
- 17) Now, try to select a state. You should see a large list of states. Unfortunately, all the states listed are American states. But that does not mean that the database does not have the states of the other countries available. It's just that since the **GetState** Aggregate has a **Max Records of 50**, we only get the first 50 states in the database, which are in this case the 50 US states. To fix this, we want to make sure we **only get the states respective to the country we choose**. And

the same criteria applies to the city, meaning that we **only want the cities that are part of the chosen State**.

Fetch States and Cities on Demand

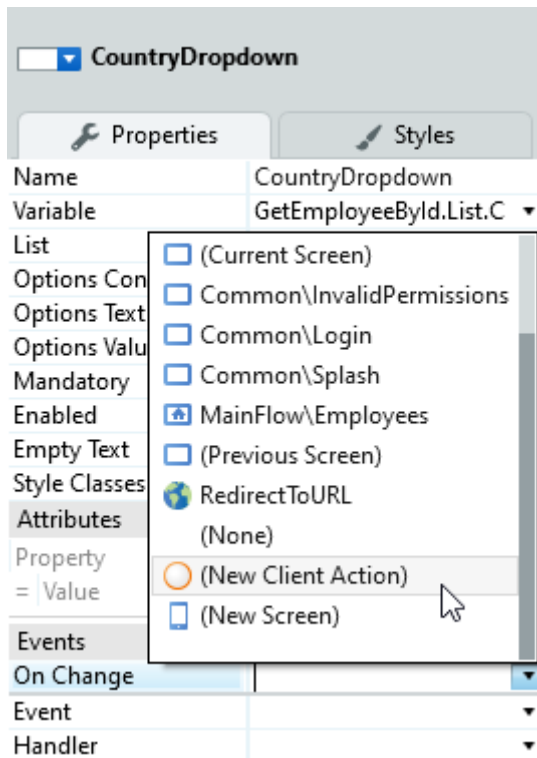
In this section, we'll solve the issue discussed above, where all states and cities (limited by the Max. Records of 50) were listed in the respective dropdowns, regardless of the country chosen. To do that, we need to change two things: first, we need to make sure that the GetStates and GetCities Aggregates are triggered **Only On Demand**, when the country and state are respectively selected; second, we need to add some **OnChange** logic to the dropdowns, so that we programmatically trigger the respective Aggregates when we choose an option in the dropdown.

- 1) Return to Service Studio and select the **GetStates** Aggregate.
- 2) Set the **Fetch** property of the Aggregate to *Only on demand*.

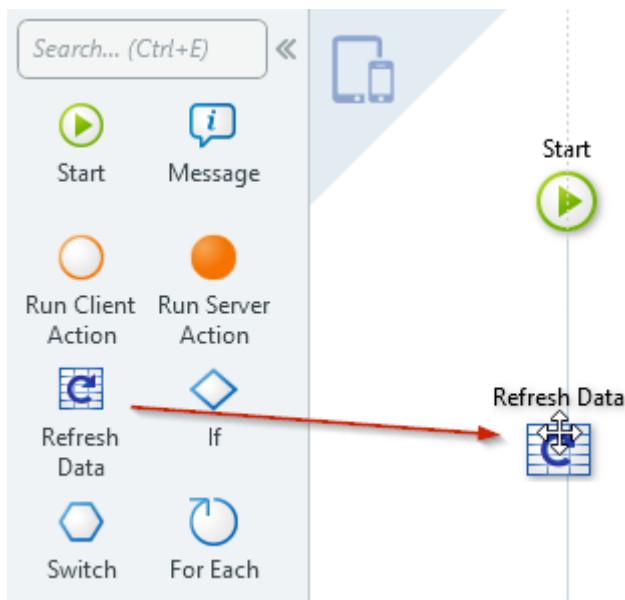


- 3) Repeat the previous step in the **GetCities** Aggregate.

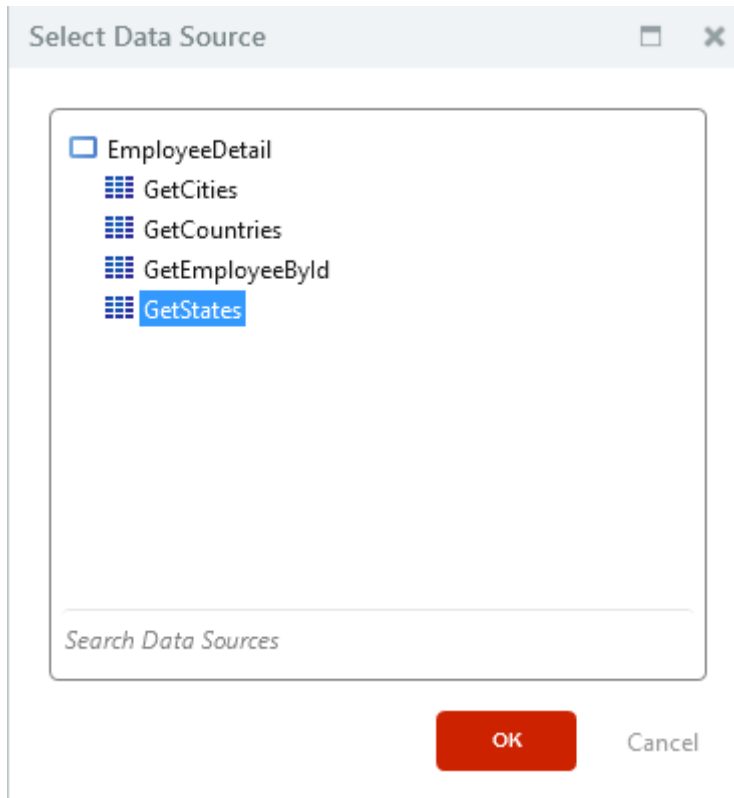
- 4) On the EmployeeDetail screen, select the **CountryDropdown** and set its **OnChange** property to *(New Client Action)*. This creates a new Action called *CountryDropdownOnChange*.



- 5) Drag a **Refresh Data** node and drop it into the action logic.

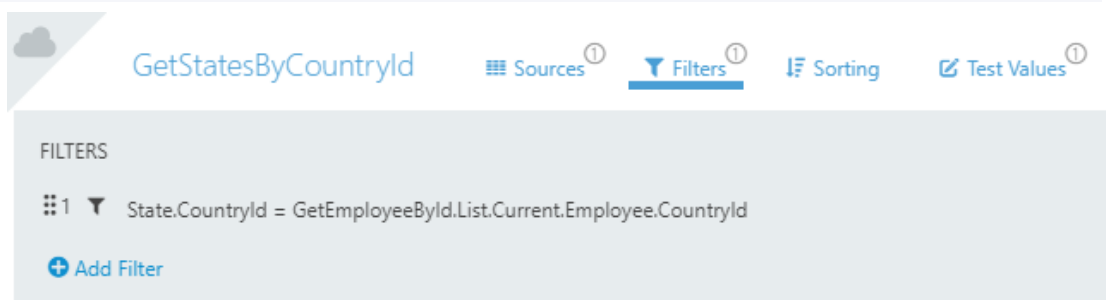


- 6) Set the Data Source to be the **GetStates** Aggregate.



- 7) To complete this logic, we just need to filter the GetStates Aggregate by the country that was selected. Open the **GetStates** Aggregate and open its **Filter** tab. Click on **Add Filter** to open the editor and add the following expression:

```
State.CountryId = GetEmployeeById.List.Current.Employee.CountryId
```



- 8) Now it's time to define a similar logic for when a state is selected. In this case, whenever the selection is made, we want to refresh the GetCities Aggregate filtered by the state. Set the **OnChange** property of the **StateDropdown** to (New Client Action). Make sure a new *StateDropdownOnChange* Action is created.

- 9) Drag a **Refresh Data** and drop it into the action logic. Select the **GetCities** Aggregate in the Data Source.



- 10) Set the **Fetch** property of the **GetCities** Aggregate to *Only on demand*.
- 11) Go to the **GetCities** Aggregate and define the following filter:
- ```
City.StateId = GetEmployeeById.List.Current.Employee.StateId
```
- 12) Publish the module and open the application in the browser.
- 13) After selecting a country, make sure that only the respective states appear. Do the same for the cities when the state is selected.
- 14) Click **Save** and then click **FetchDataOnDemandExcercise** on the top left to return to the Employee List. Click the record you changed and notice that the State and city fields are not correct and the drop down does not contain options. You will need to use the **On After Fetch** event to re-run the action that updates those fields.
- 15) Return to the module and click on the **GetCountries** aggregate. In the properties, set the **On After Fetch** event to *CountryDropdownOnChange*. Then go to the **GetStatesByCountryID** and set the **On After Fetch** event to *StateDropdownOnChange*.
- 16) Publish the module and open the application in the browser. The record you changed should perform correctly.