Appointment Diagnosis

Table of Contents

Scenario **2**
Actions 3

How to **4**
Patient Situation Area 4
PatientHealthStatus Entity 6
Health Status Buttons 12
Diagnosis Text and Save Button 15
Save Appointment 20
Form Not Valid 20
AppointmentDetail_Update Action 21
Create the Logic 23
Finish up 28
Testing and Results 31

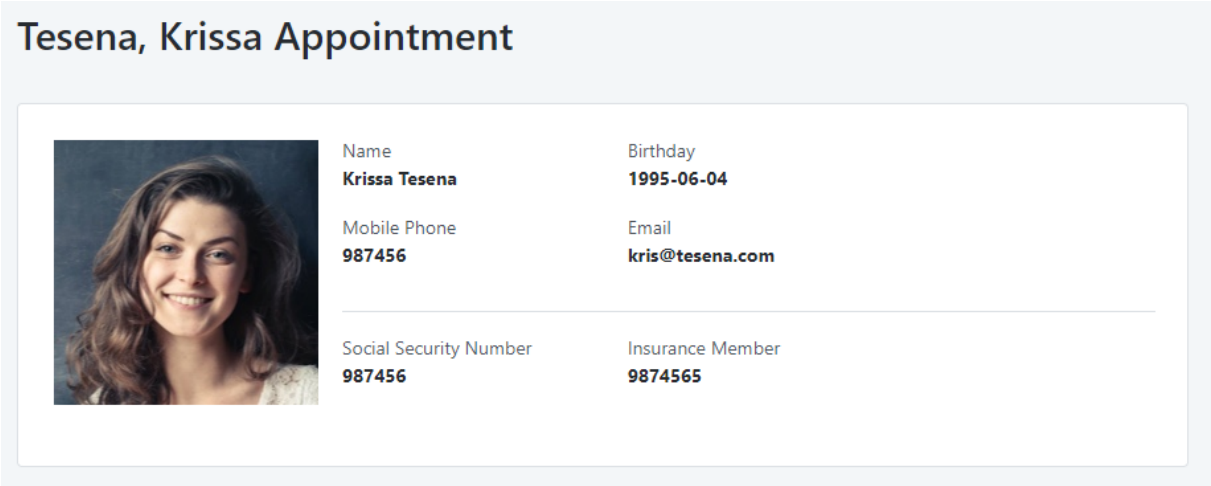
Wrapping up **33**
References 33

knowledge@outsystems.com

www.outsystems.com

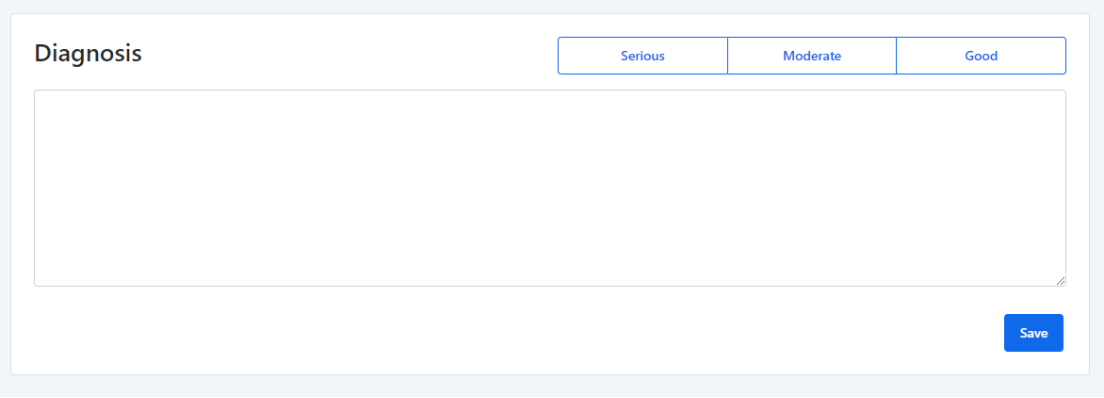
Scenario

In the previous tutorial, you started creating the Appointment_Detail Screen, with the information about the patient.



Now, you will add an extra Form with a few functionalities:

- Patient health status: the Doctor will be able to select if the patient's health status is good, moderate or serious, by choosing one of three Buttons.
- Diagnosis Text: a text area for the doctor to write the patient's diagnosis.
- Save: a Save button for the doctor to submit the health status and the diagnosis, and set the appointment as complete.



Actions

To make sure all of this works, you will need a few different Actions. These Actions will have the logic to update the patient and appointment information.

- Update Appointment's Detail - Server-side Action that will update the Patient's health status and the status of the appointment in the database.
- Save Appointment Detail On Click - Client-side Action that is triggered when the Doctor clicks on the Save Button. This Action will also validate if the Form's fields were filled in. If not, it won't let the Doctor save the Appointment Detail. If they are, the Update Appointment's Detail Action will be called to update the database.

How to

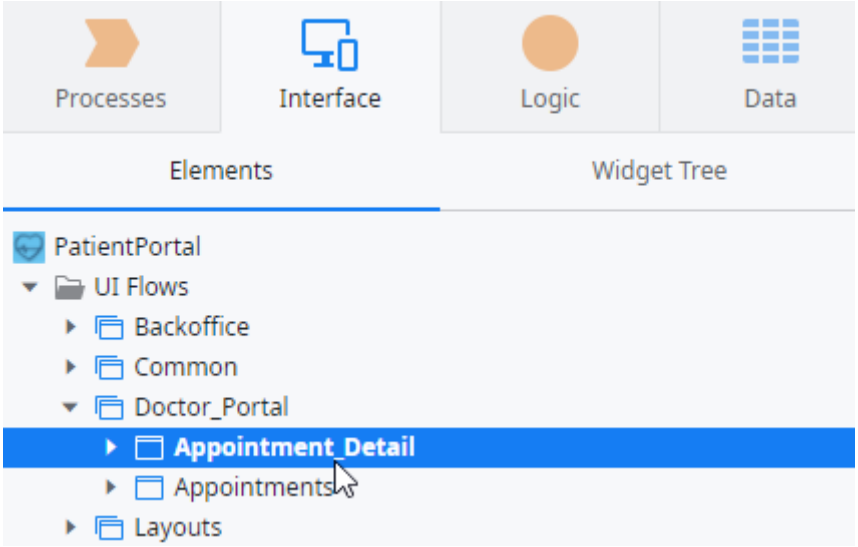
Now that you know the scenario, let's build those features in the Patient Portal app by following this how-to guide! Are you ready? Let's do it together!

Patient Situation Area

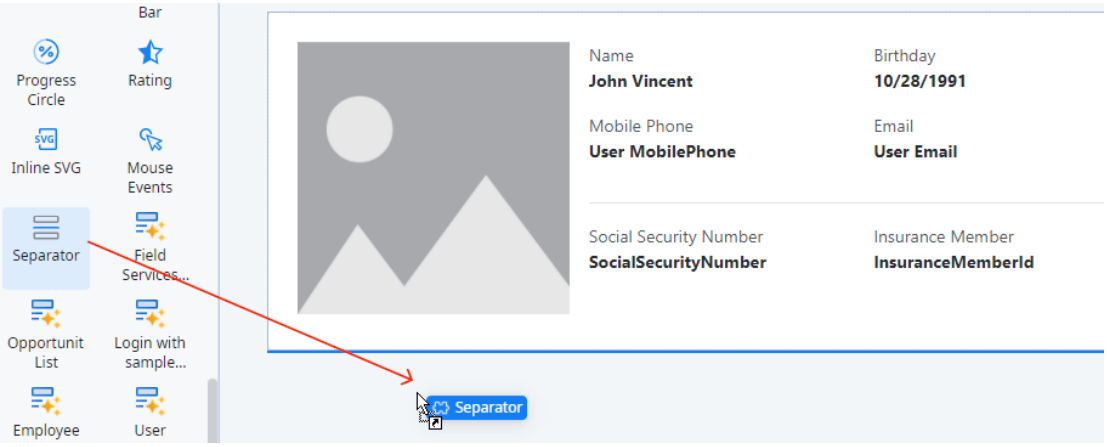
You will now have an area on this Screen where the doctor will be able to write a diagnosis and determine the health status of the patient.

For that, let's start by adding some elements to help us make the UI look nice.

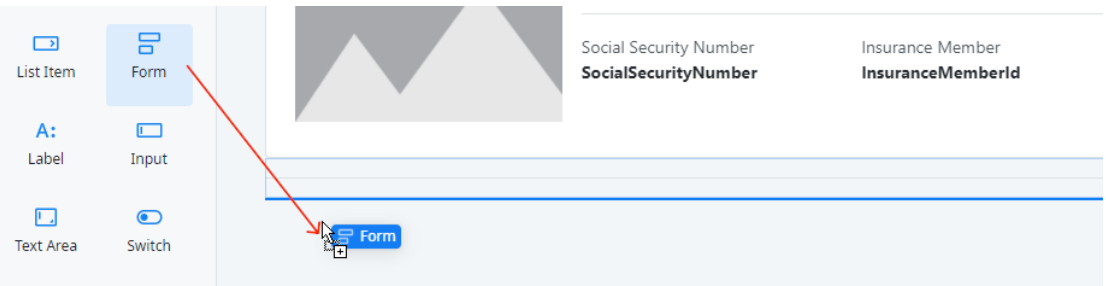
1) Open the **Appointment_Detail** Screen, if not already open.



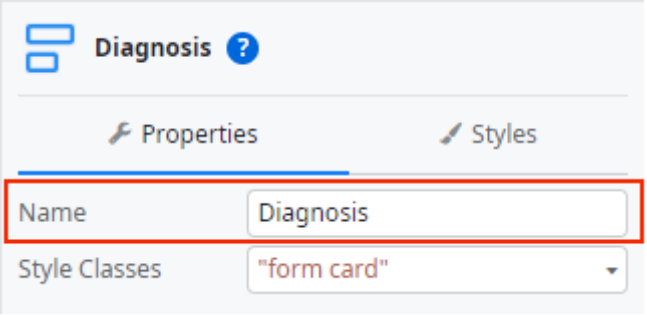
2) Drag a **Separator** from the left sidebar and drop it under the Card.



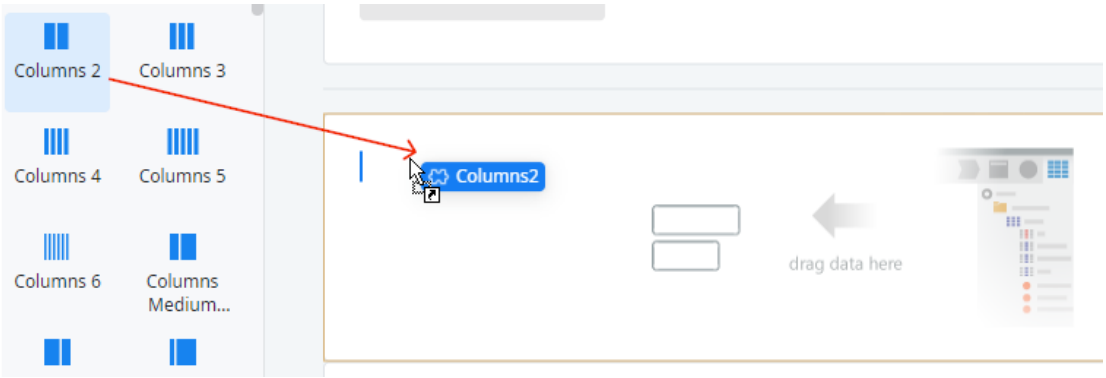
3) Drag a **Form** from the left sidebar and drop it under the Separator.



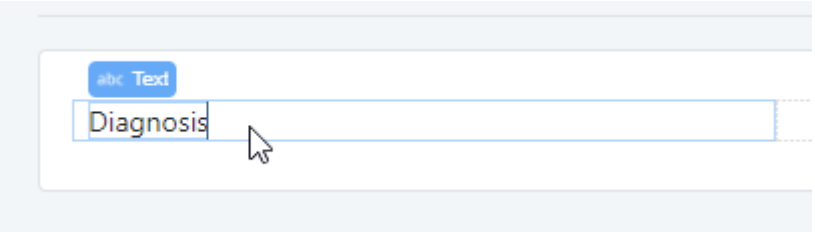
4) Click on the Form to open its properties and set the **Name** to *Diagnosis*.



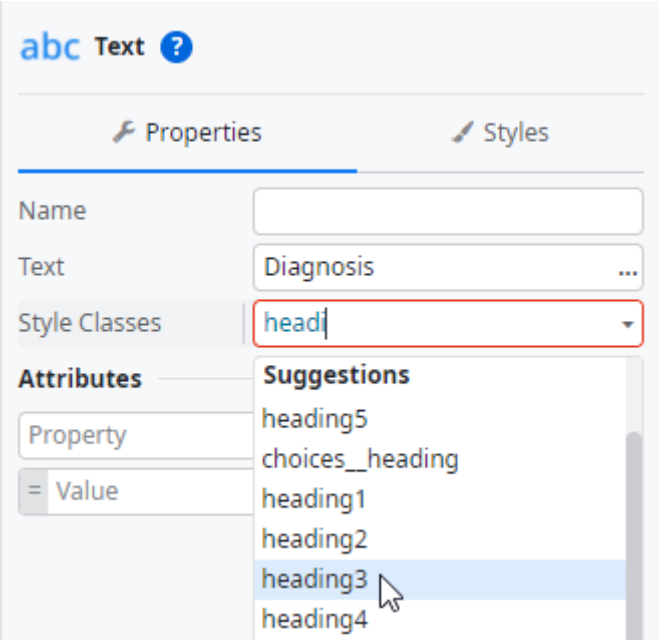
5) Drag and drop a **Columns2** inside the Form.



6) Click on the first column and type *Diagnosis*.

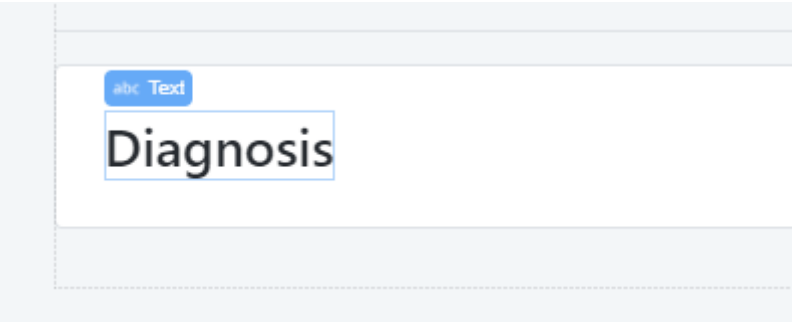


7) In the Text's properties, select the **Style Class** *heading3* in the dropdown.



Tip: you can search for Style classes by typing in the dropdown of the property.

This is what your text will look like after the Style Class is added:



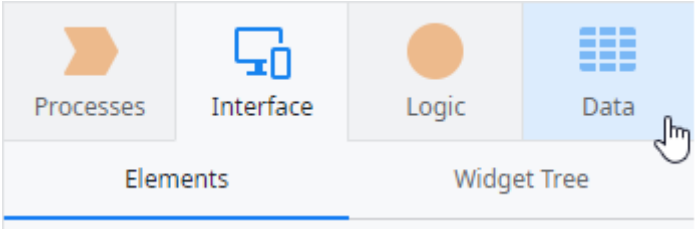
PatientHealthStatus Entity

To classify the patient's health status, you need to create a new Entity called *PatientHealthStatus*.

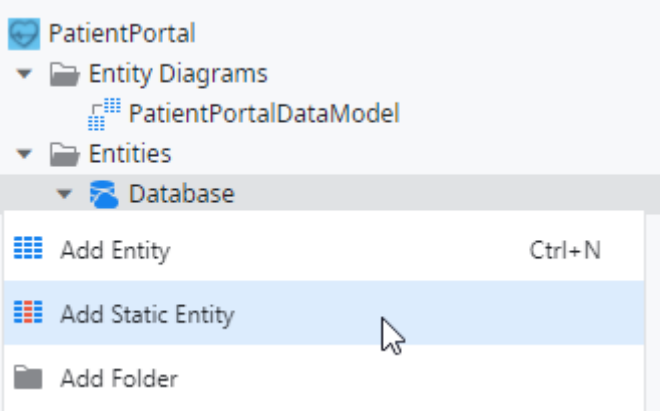
The Patient's health status can have different values, such as "Serious", "Moderate", and "Good". Since these values are limited to the ones we already know, you can use a

Static Entity. A Static Entity consists of a set of named values that cannot be changed during execution time.

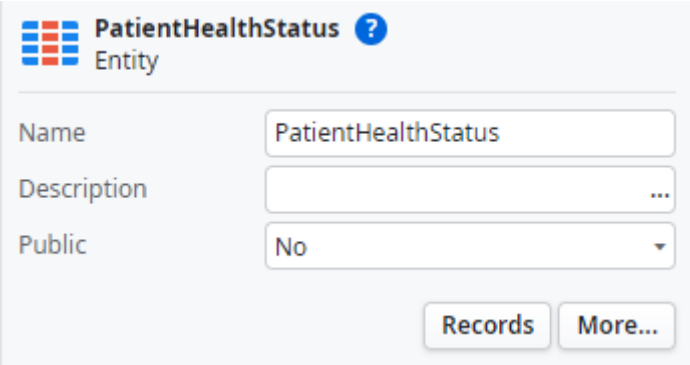
1) Click on the **Data** tab.



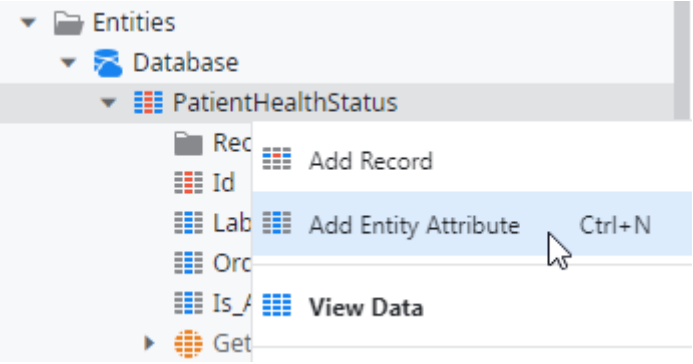
2) Right-click on the Database and select **Add Static Entity**.



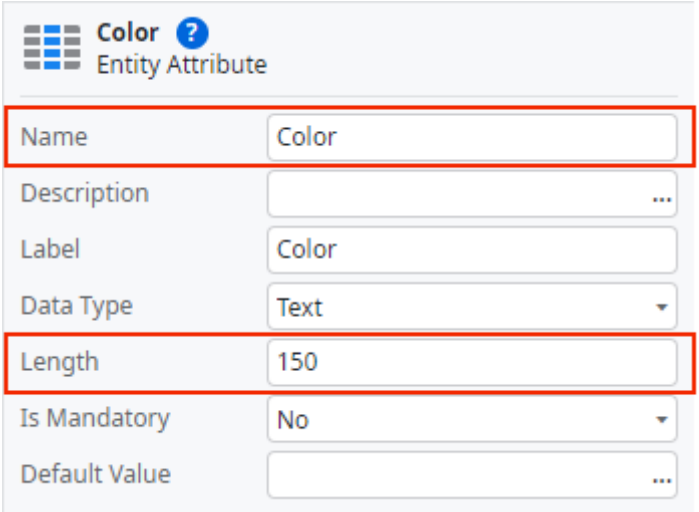
3) Name the Entity *PatientHealthStatus*.



4) Right-click on the new Entity and select **Add Entity Attribute**.

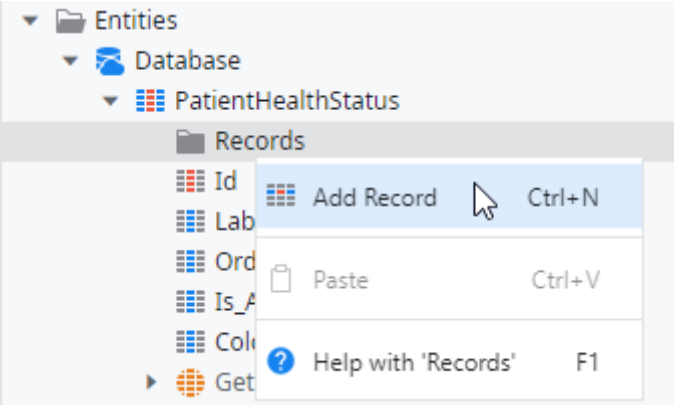


5) Set the **Name** of the attribute to *Color*, keep the **Data Type** as **Text**, and change the **Length** to *150*.



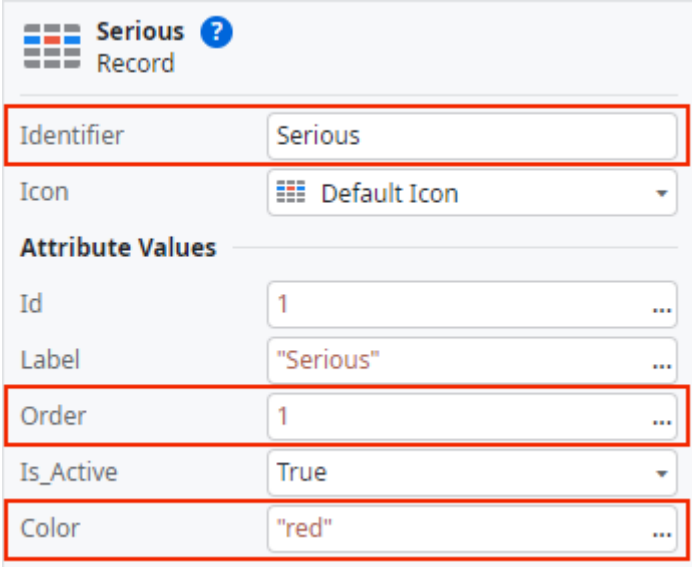
Now that the Entity has all the Attributes you need, let's add the records, which are the possible values for the health status (Serious, Moderate and Good).

6) Right-click on the **Records** folder inside the Static Entity and select **Add Record**.



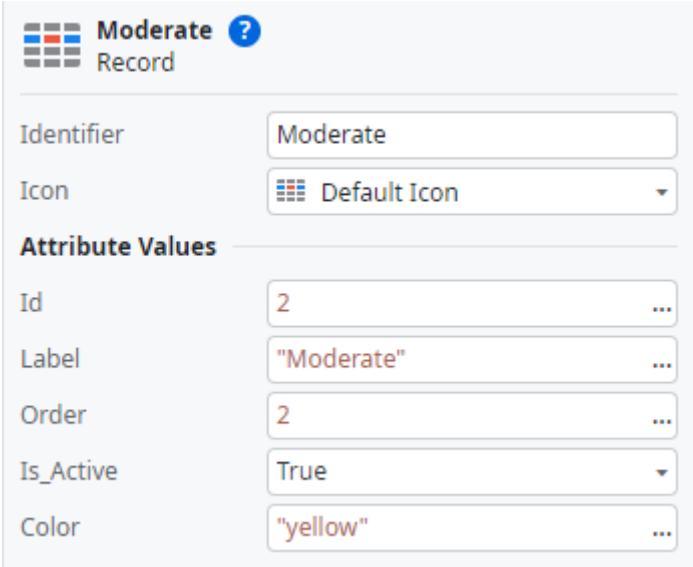
For each value, you will need to add the Label, Order, and Color.

7) For the first record, set the **Identifier** to *Serious* and the **Color** to *"red"* in the Color.



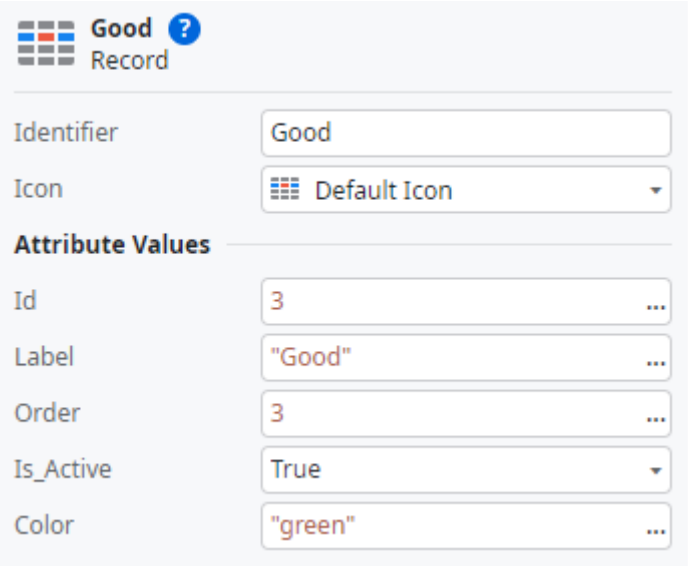
8) Repeat the same steps for the **Moderate** status.

9) Set the **Identifier** to *Moderate* and the **Color** to *"yellow"*.



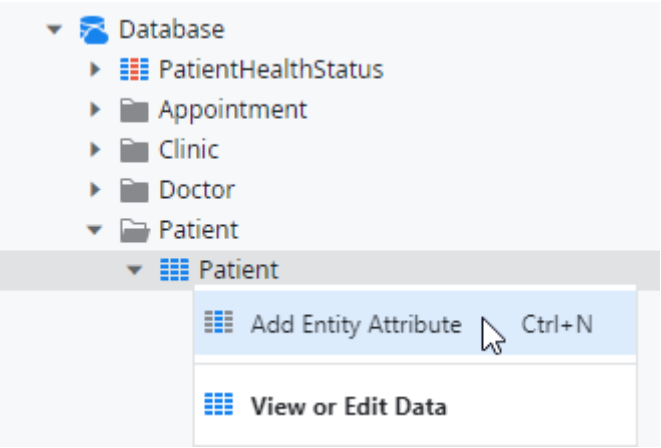
10) Repeat the same steps for the **Good** status.

11) Set the **Identifier** to *Good* and the **Color** to *"green"*.

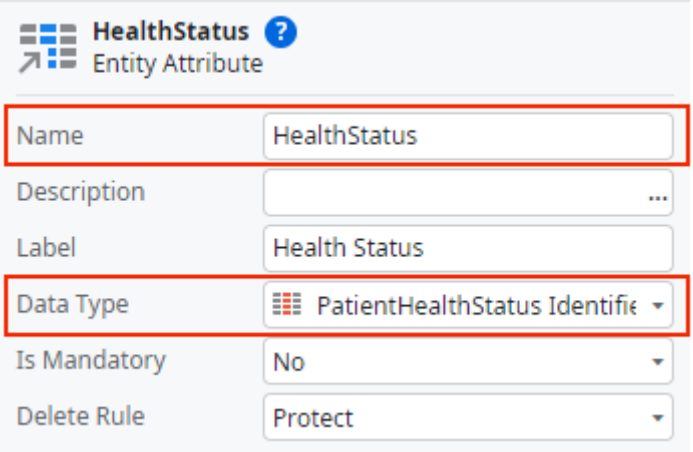


Now we need to add a new attribute to the Patient Entity to store the Patient's health status.

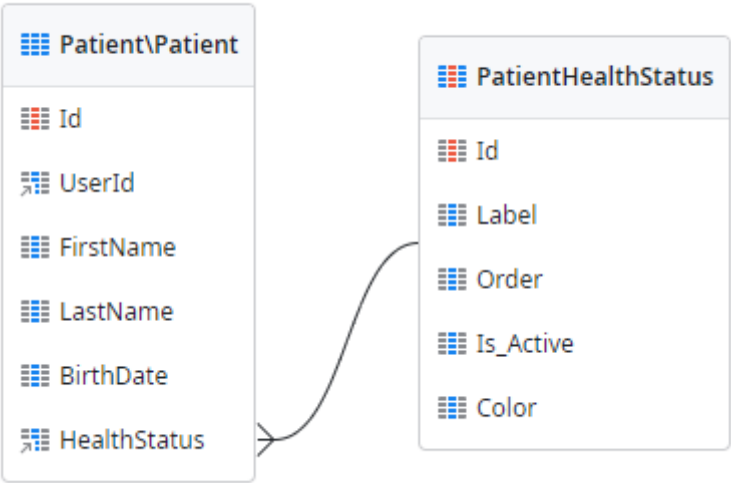
12) Expand the **Patient** folder, then right-click on the **Patient** Entity and select **Add Entity Attribute**.



13) Set the **Name** to *HealthStatus* and the **Data Type** to **PatientHealthStatus Identifier**.



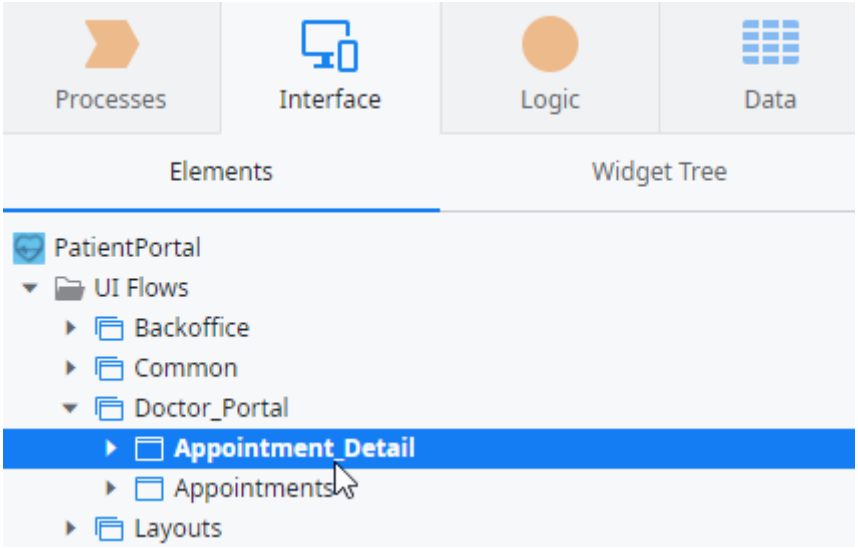
In summary, the Patient Entity now has an attribute linked to the PatientHealthStatus Entity.



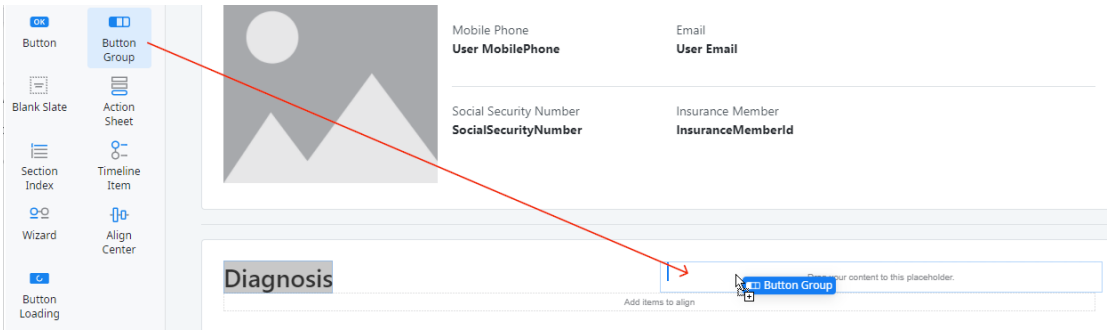
Health Status Buttons

Now that your data model was updated, you can use the new PatientHealthStatus Entity to create three Buttons, one for each status. The doctor will then use them to define the patient's status.

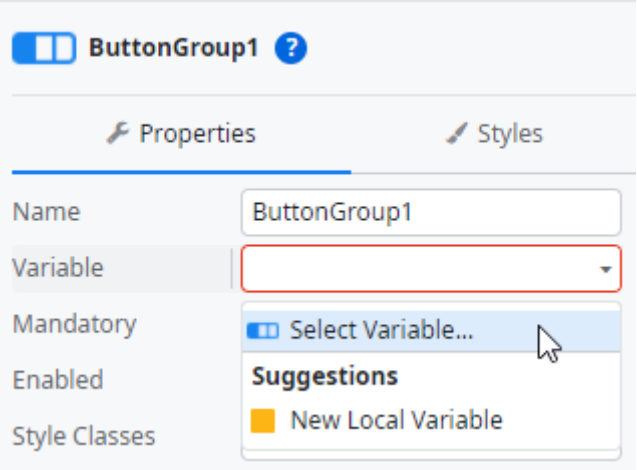
1) Go back to the **Appointment_Detail** Screen by double-clicking on it.



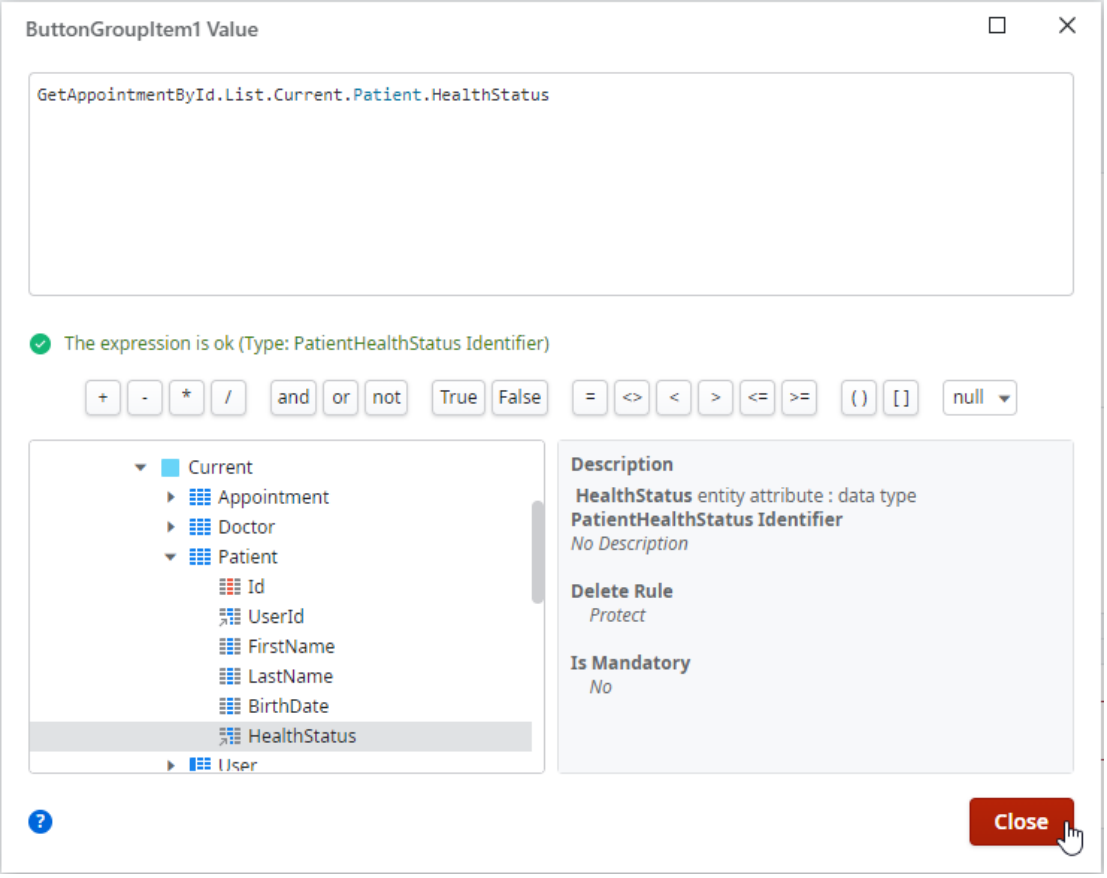
2) Drag a **Button Group** element from the left sidebar to the second column.



3) In the ButtonGroup's properties, click on the dropdown of the **Variable** property and select **Select Variable**.



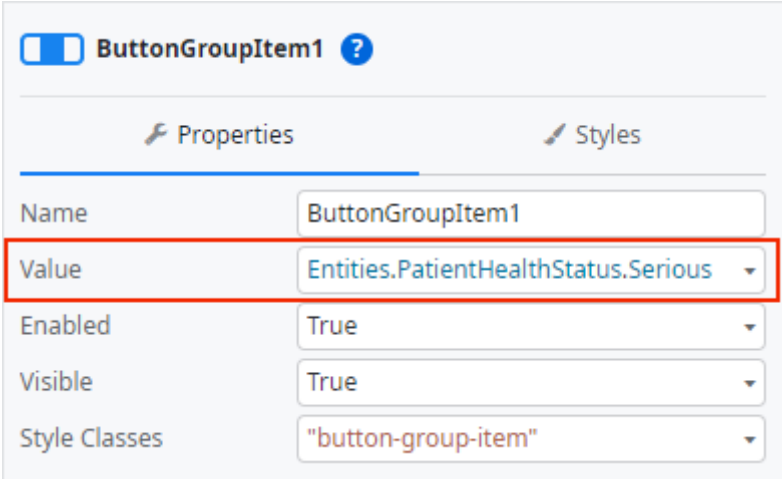
4) In the Value dialog, expand the GetAppointmentById Aggregate, then List, Current, Patient and then select the **Health Status** with a double-click. You can also simply type `GetAppointmentById.List.Current.Patient.HealthStatus`.



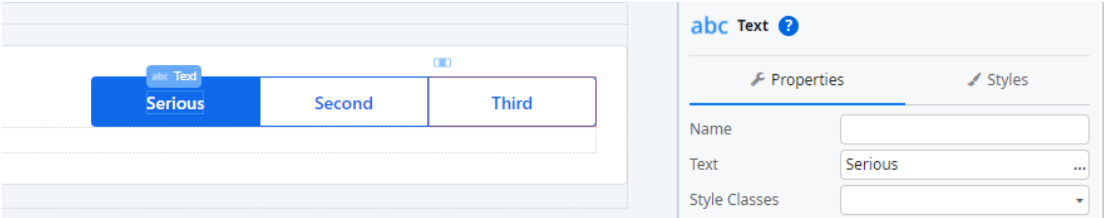
Here, you are tying together the new attribute that you just added to the Patient Entity, with the option that the doctor will make for the health status.

Now, you need to setup the value that each button from the group will have (Serious, Moderate and Good).

5) Click on the **first** Button Item, then click on the dropdown of the **Variable** property and select **Serious**. It will automatically change the value to **Entities.PatientHeathStatus.Serious** .

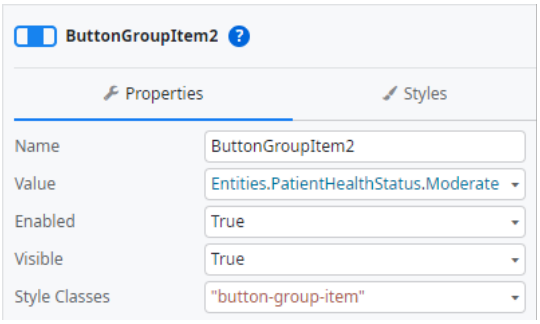


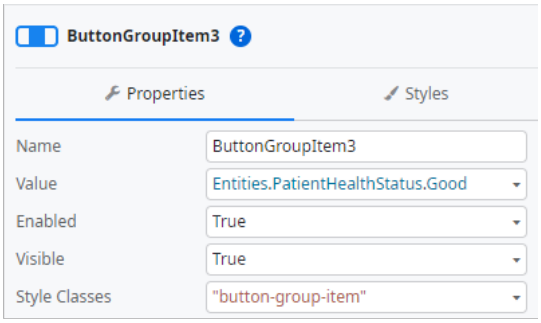
6) Click on the text inside the first Button in the Screen preview, then type *Serious*.



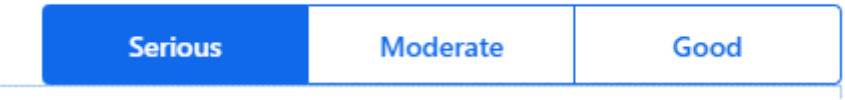
Now, let's repeat the same process for the Moderate and Good statuses, for the second and third buttons respectively.

7) Set the **Value** of the second and third buttons to *Moderate* and *Good* respectively, just like you did for the first button.





8) In the Screen preview, replace the second and third button texts by *Moderate* and *Good*.

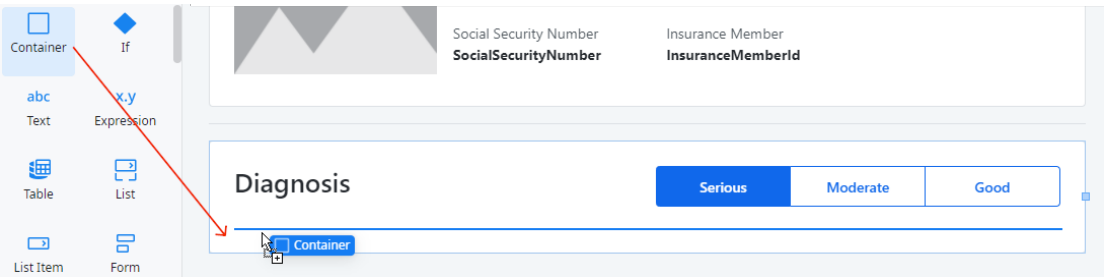


Diagnosis Text and Save Button

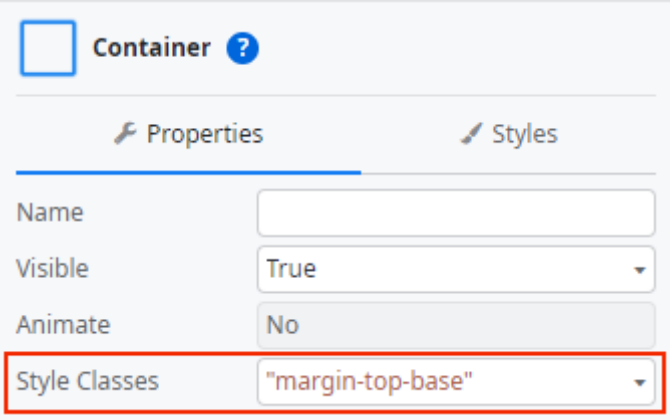
Besides the health status, the doctor will also be able to write a diagnosis, in a text area. This information will be stored and be available for Doctors and Patients afterward.

You will also add a Save button that will trigger the logic to save the appointment.

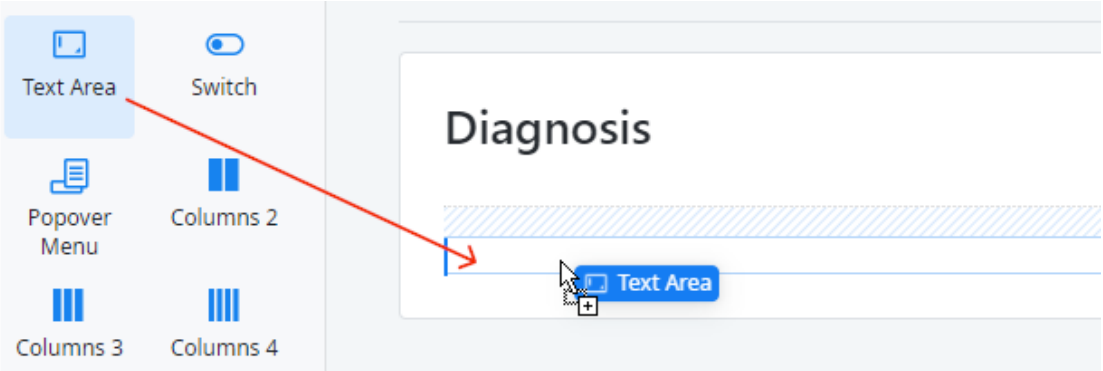
1) Still in the **Appointment_Detail** Screen, drag and drop a **Container** inside the Form, right after the Columns.



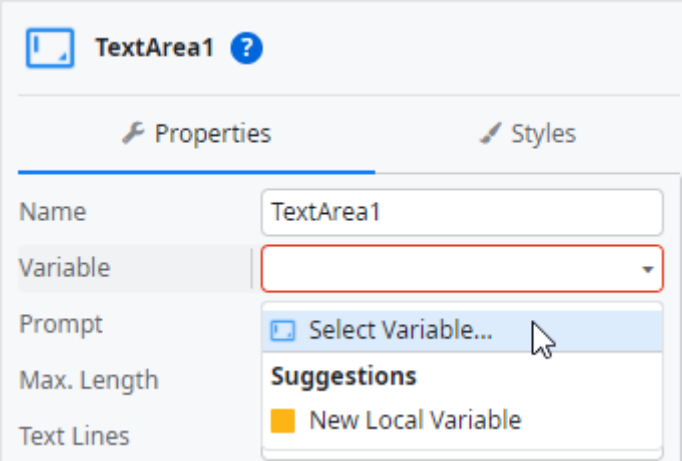
2) Set the **Style Classes** property to `"margin-top-base"` to the Style Classes.



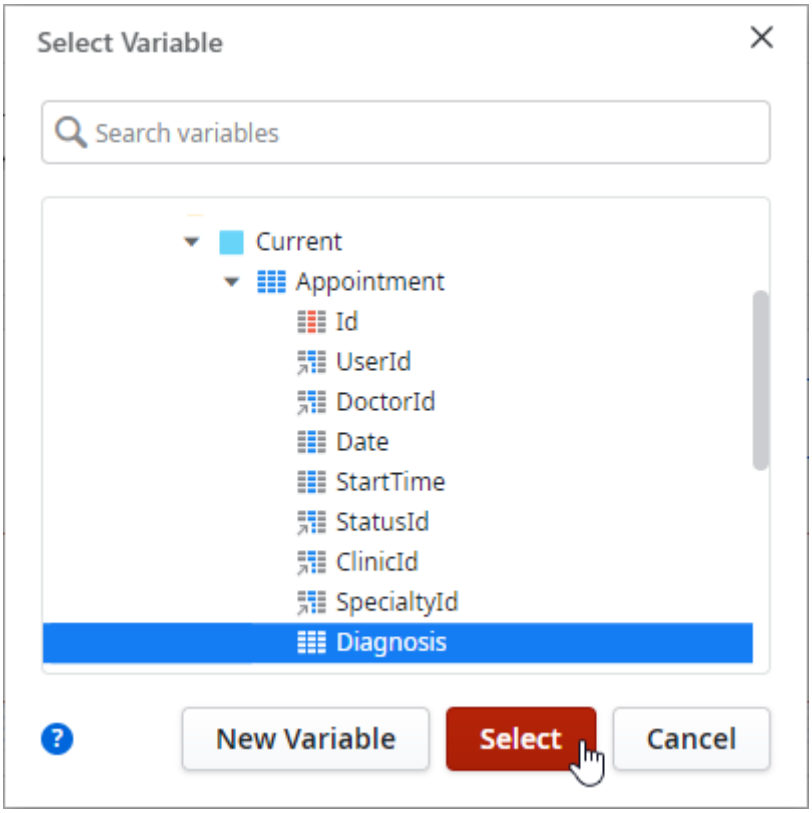
3) Drag and drop a **Text Area** element inside the Container.



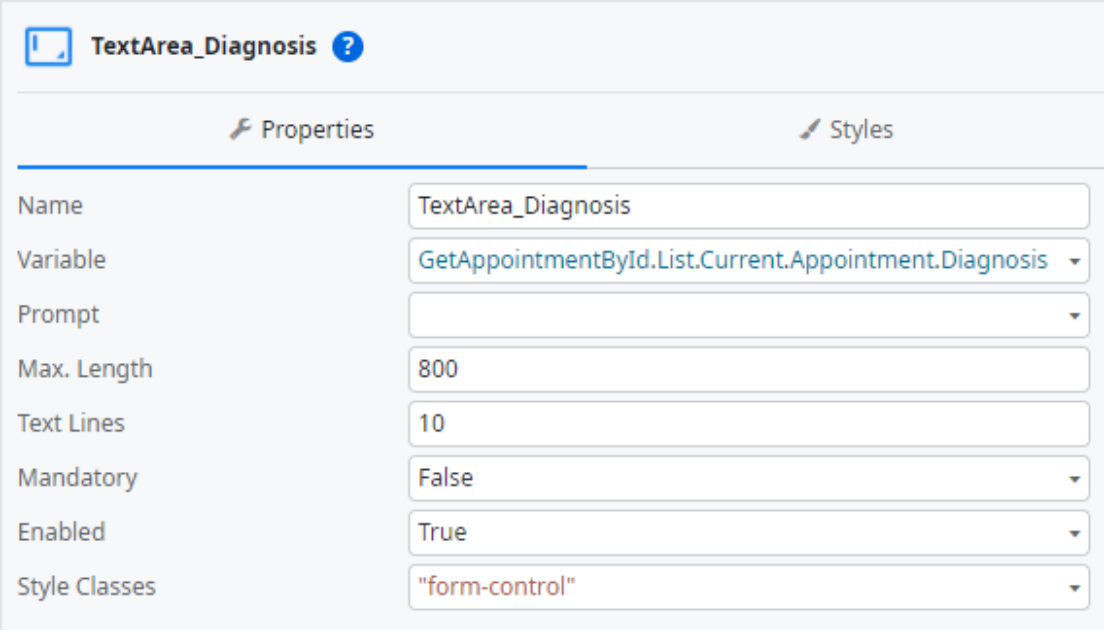
4) Click on the **Variable** property of the Text Area and click on **Select variable**.



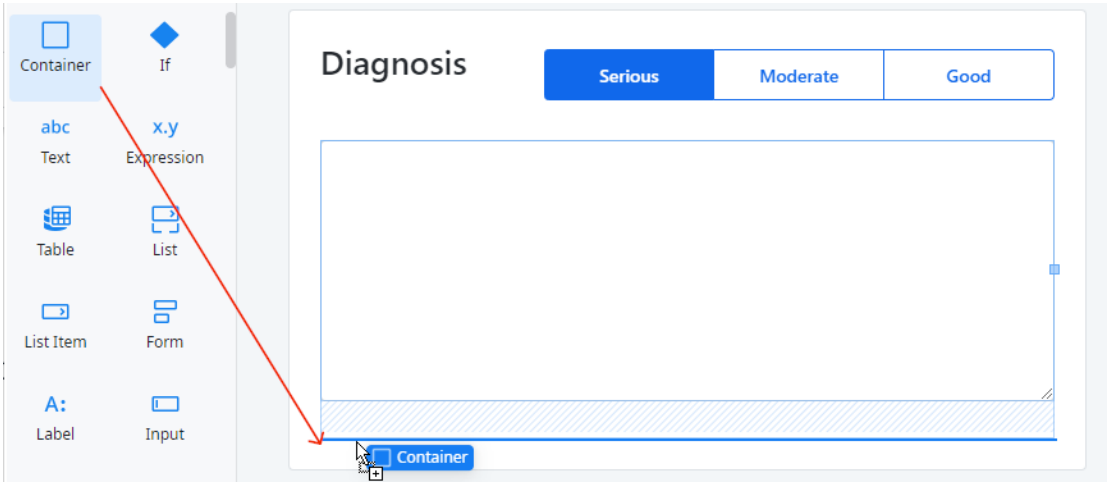
5) Expand the **GetAppointmentById** Aggregate until you find the **Appointment** Entity. Then, select the **Diagnosis** attribute.



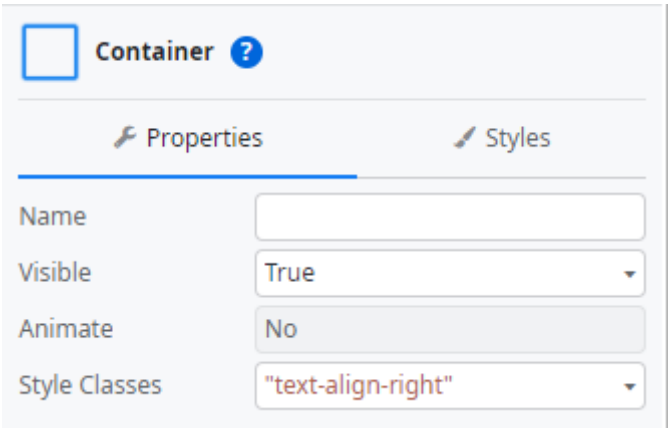
6) Still in the Text Area properties, keep the **Max. Length** as *800* and change the **Text Lines** to *10*.



7) Drag and drop another **Container** under the previous one.

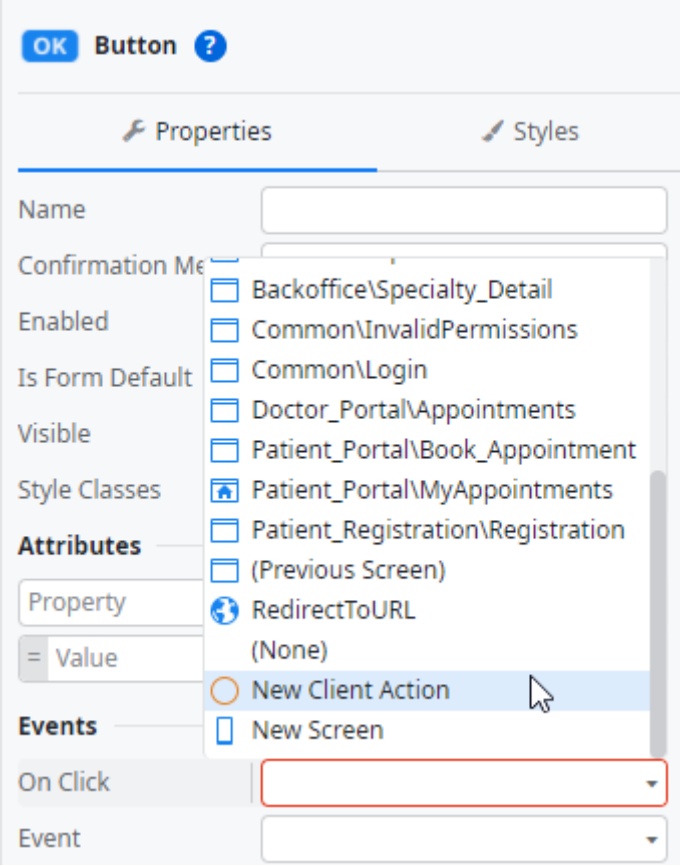


8) Set the **Style Classes** property to `"text-align-right"`.

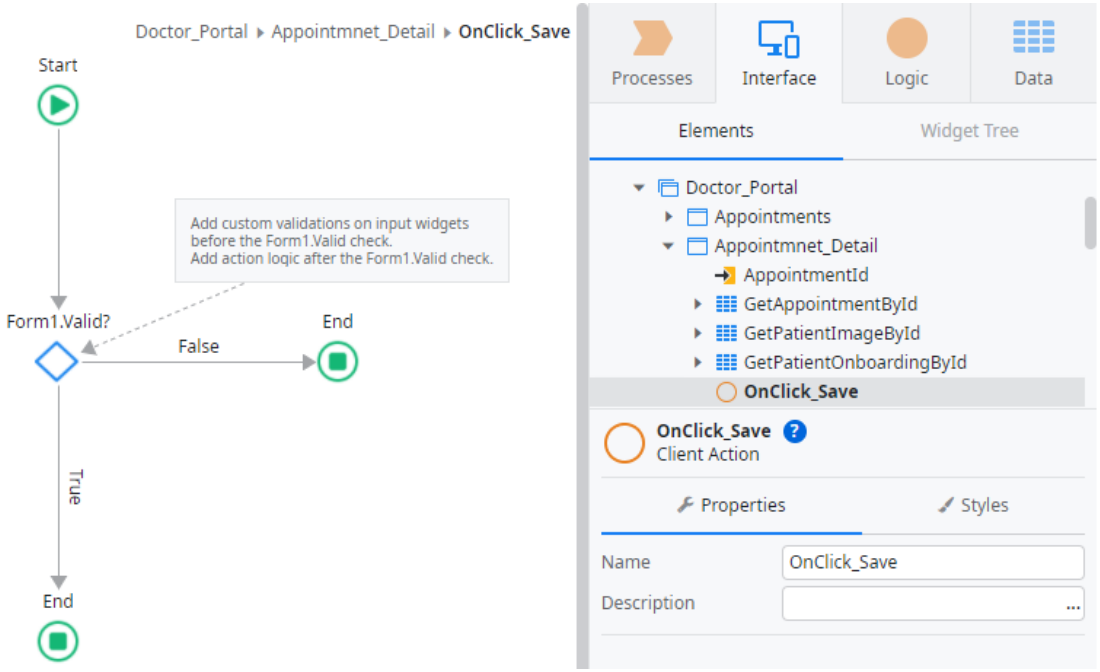


9) Drag and drop a **Button** inside the Container.

10) Click on the Button to open its properties. Set the **OnClick** property to **New Client Action**.



11) Set the **Name** of the Action to *OnClick_Save*

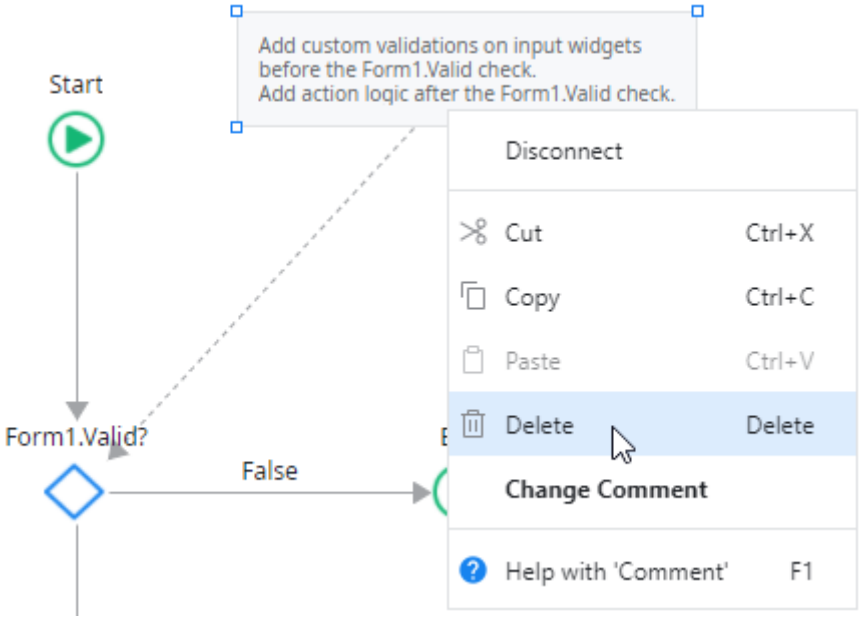


Save Appointment

The OnClick_Save Action will have the logic to save the Appointment with the Doctor's diagnosis. But first, it needs to validate if all the mandatory fields were filled out. Fortunately, OutSystems already set that up for you. You just need to define what happens when the Form is not valid.

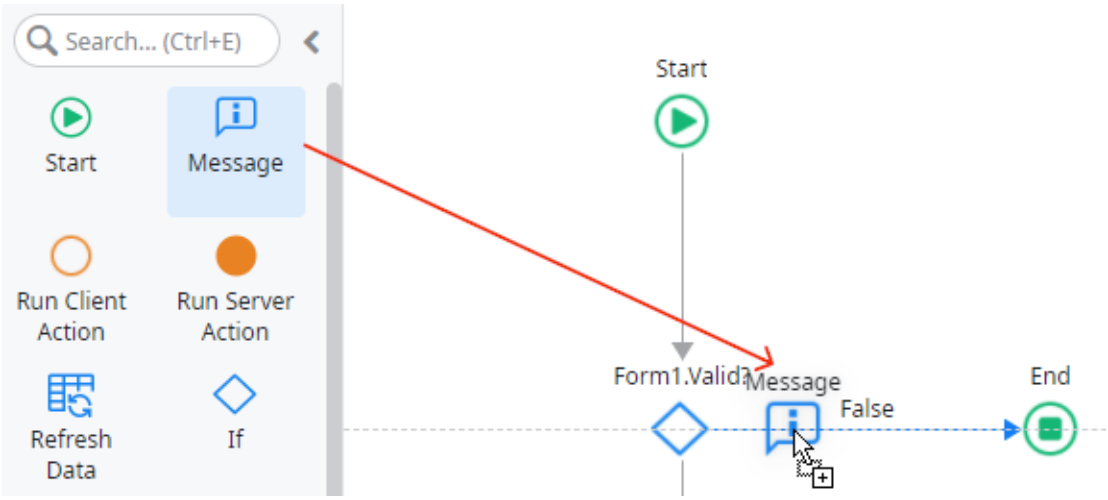
Form Not Valid

1) Still in the OnClick_Save Action, right-click on the **Comment** and select delete to remove it.

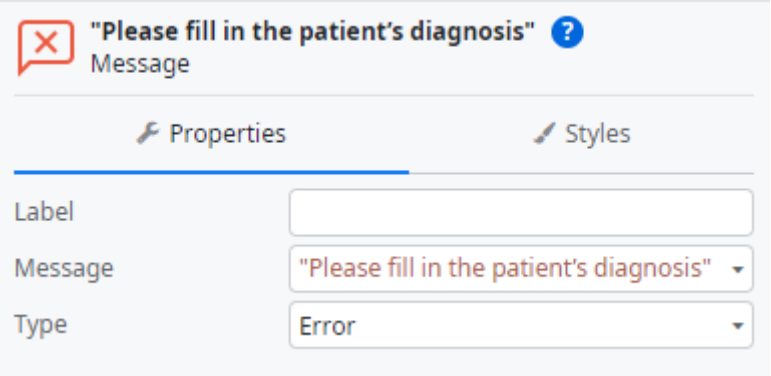


Comments can be very useful to help you or other developer understand the Action, but it does not affect the Action's behavior!

2) Drag a **Message** from the left sidebar and drop it in the False branch.



3) In the Message properties, type the message *"Please fill in the patient's diagnosis."* and change the Type to **Error**.

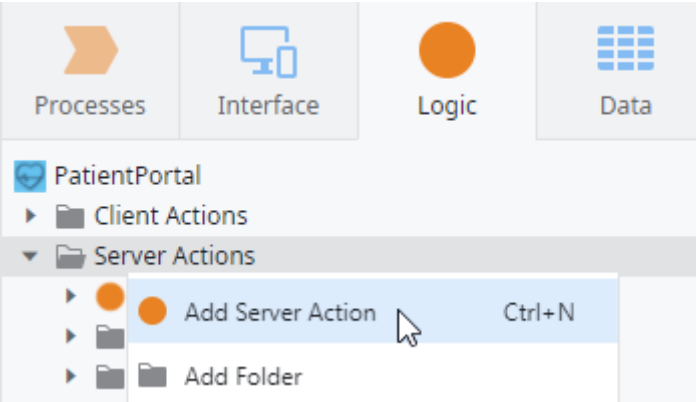


Now, for the True branch, you will need to define the logic to write the changes in the database. You need a new Server Action to help you with that.

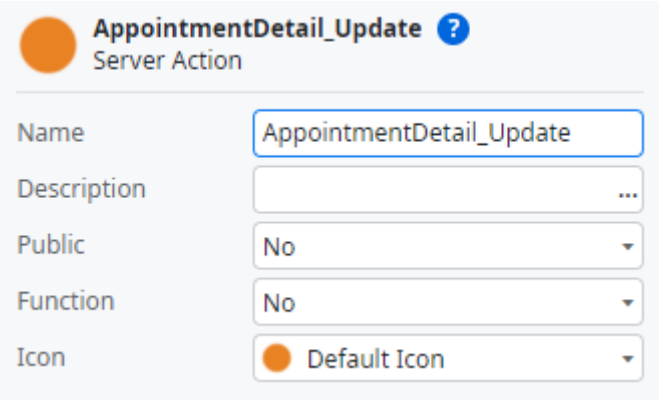
AppointmentDetail_Update Action

You will create a new Action that updates the patient information in the database, as well as the status of the appointment. Let's start by defining the parameters of the Action.

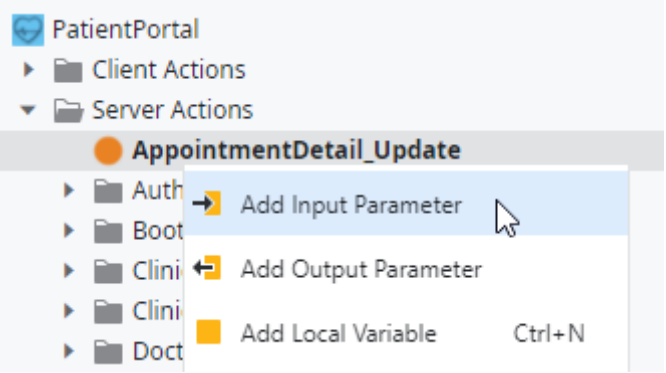
1) Open the Logic tab. Right-click on the **Server Actions** folder and select **New Server Action**.



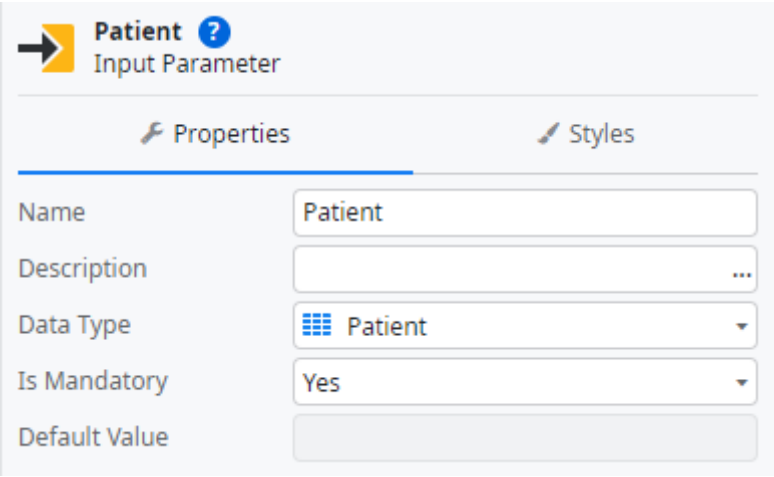
2) Set the **Name** of the Action to *AppointmentDetail_Update*.



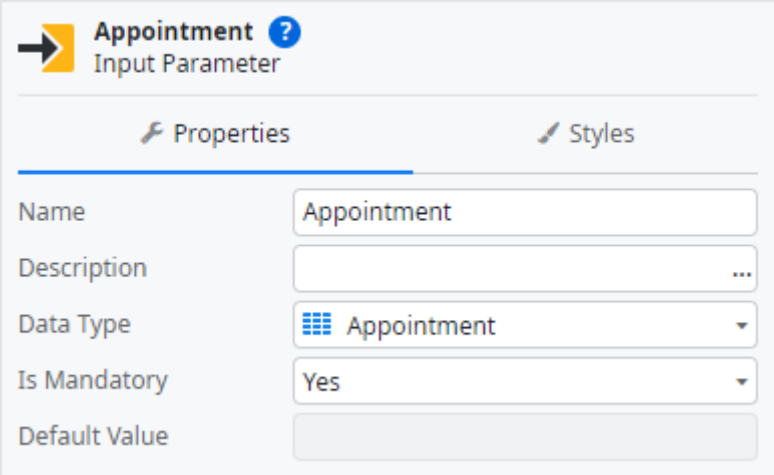
3) Right-click on the Action and select **Add Input Parameter**.



4) Set the **Name** of the Input to *Patient*. The **Data Type** will change to **Patient**.



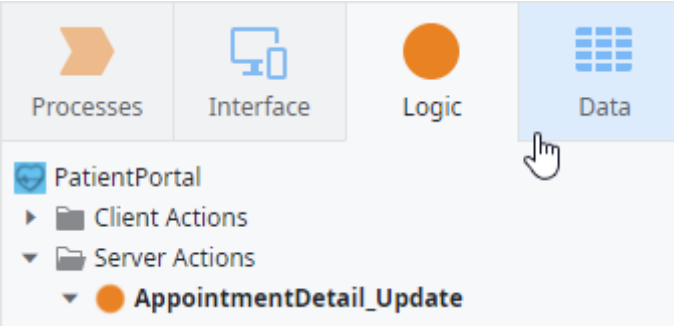
5) Add another Input Parameter, set its **Name** to *Appointment* and make sure the **Data Type** is set to **Appointment**.



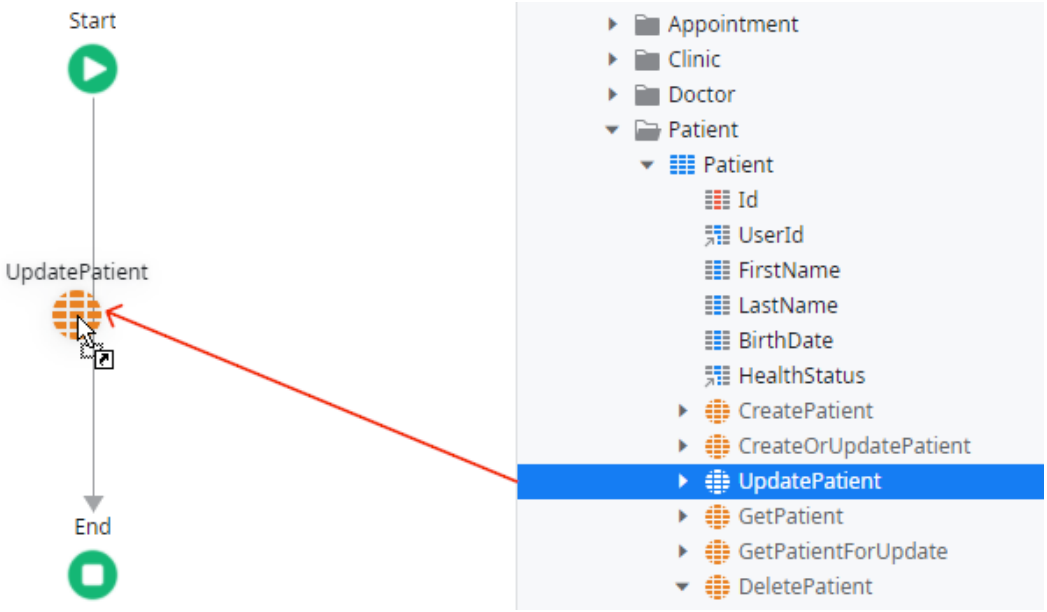
Create the Logic

Let's now define the logic of this Action. This logic should update the patient, set the appointment status to complete and then update the appointment as well in the database.

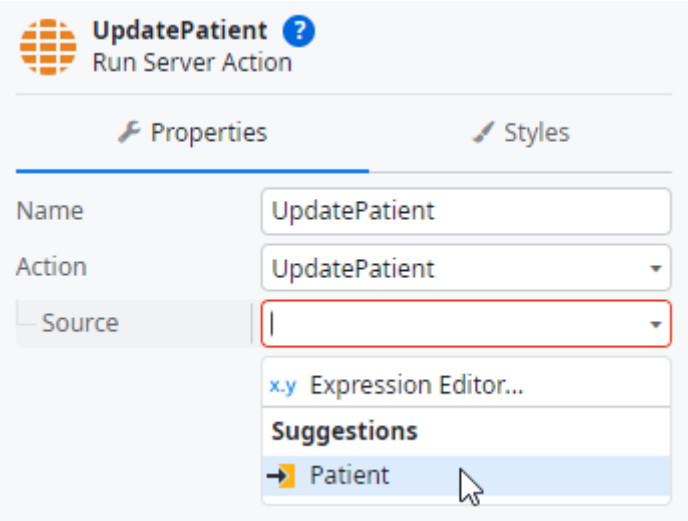
1) Click on the **Data** tab.



2) Expand the **Patient** Entity and select the **UpdatePatient** Action. Drag and drop it in the AppointmentDetail_Update Action Flow.

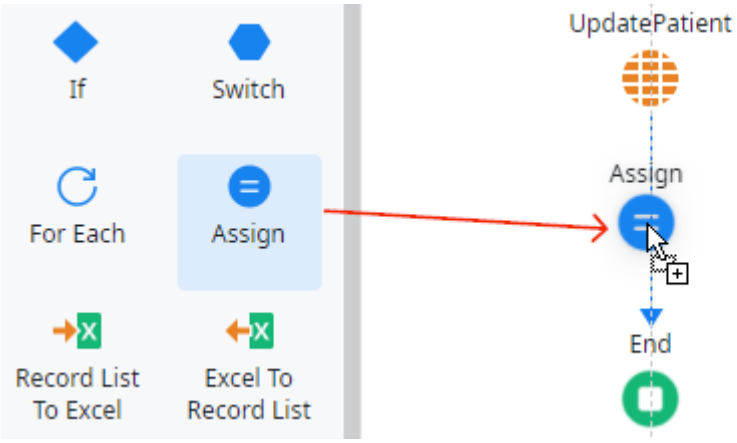


3) Select the **Patient** Input Parameter as the Source.

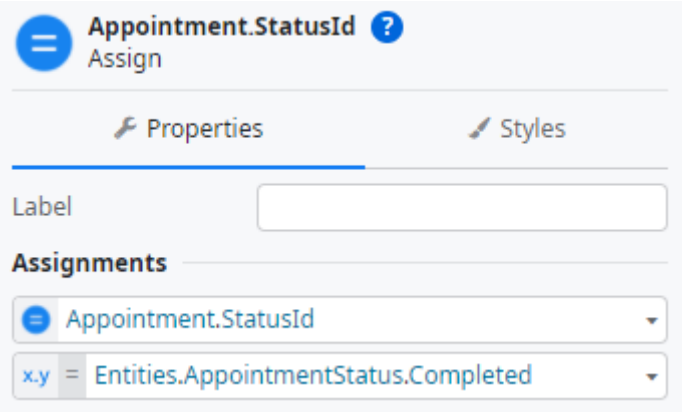


When the Start Button is clicked, the Appointment's status must change from Submitted to Completed. We should use an Assign to accomplish that.

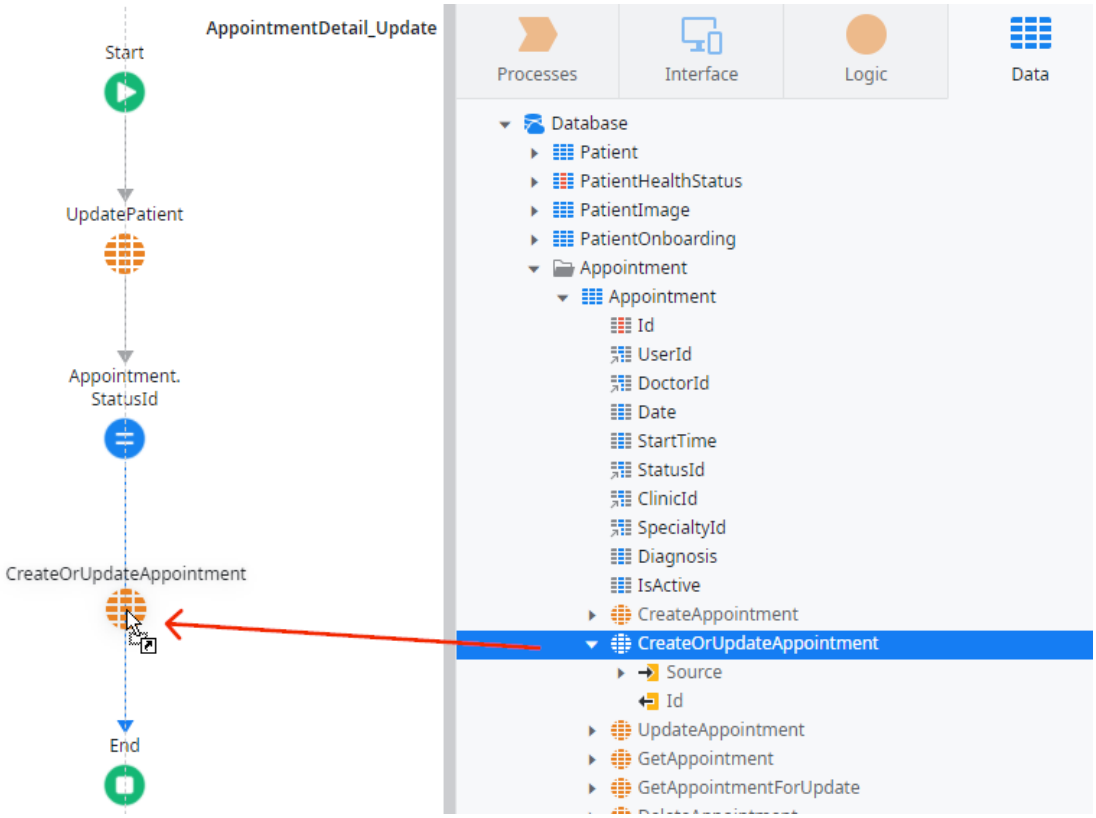
4) Drag an **Assign** from the left sidebar and drop it under the UpdatePatient Action in the flow.



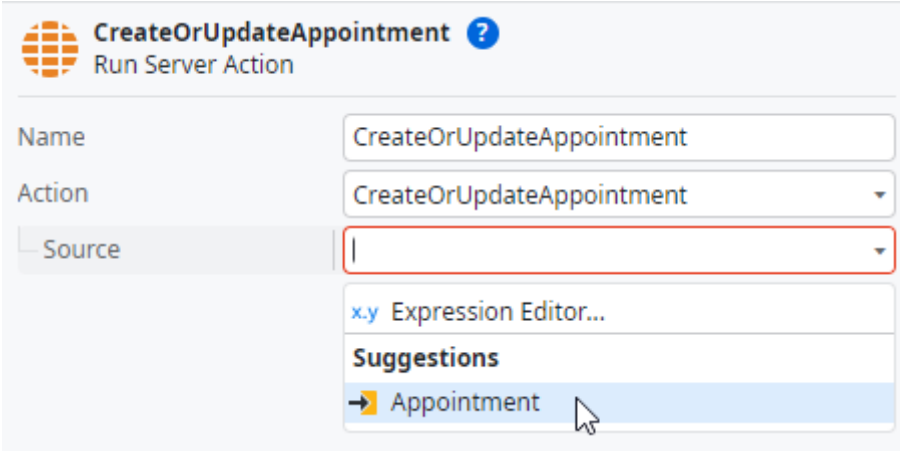
5) Set the **Variable** to **Appointment.StatusId** and the **Value** to **Entities.AppointmentStatus.Completed**.



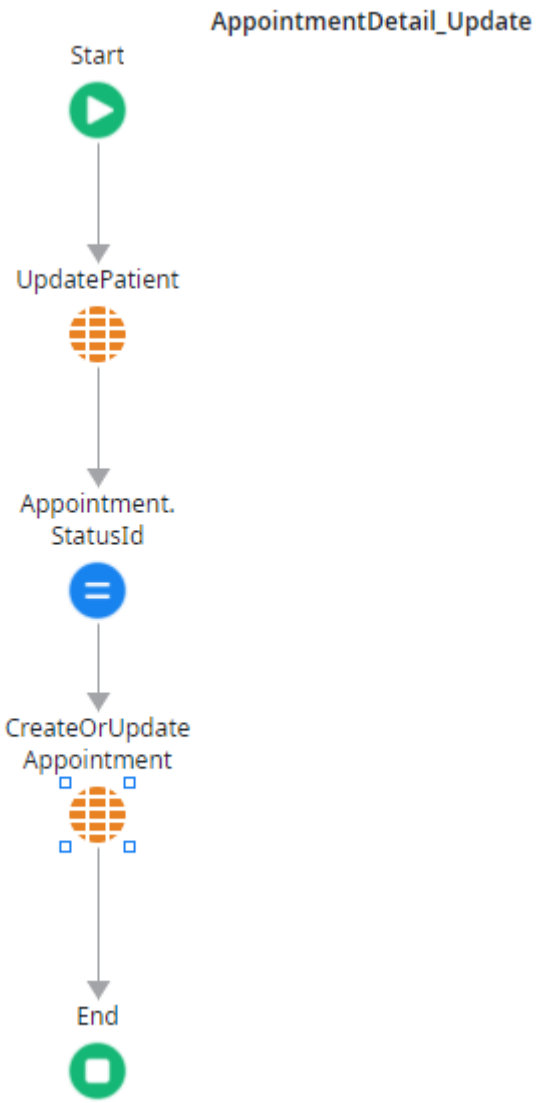
6) Still in the **Data** tab, expand the **Appointment** Entity. Drag the **CreateOrUpdateAppointment** Action and drop it right after the Assign.



7) Set the **Appointment** value to the **Appointment** Input Parameter of the Action.



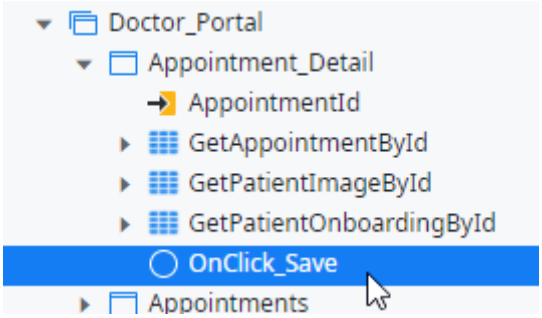
This is what the Action will look like in the end:



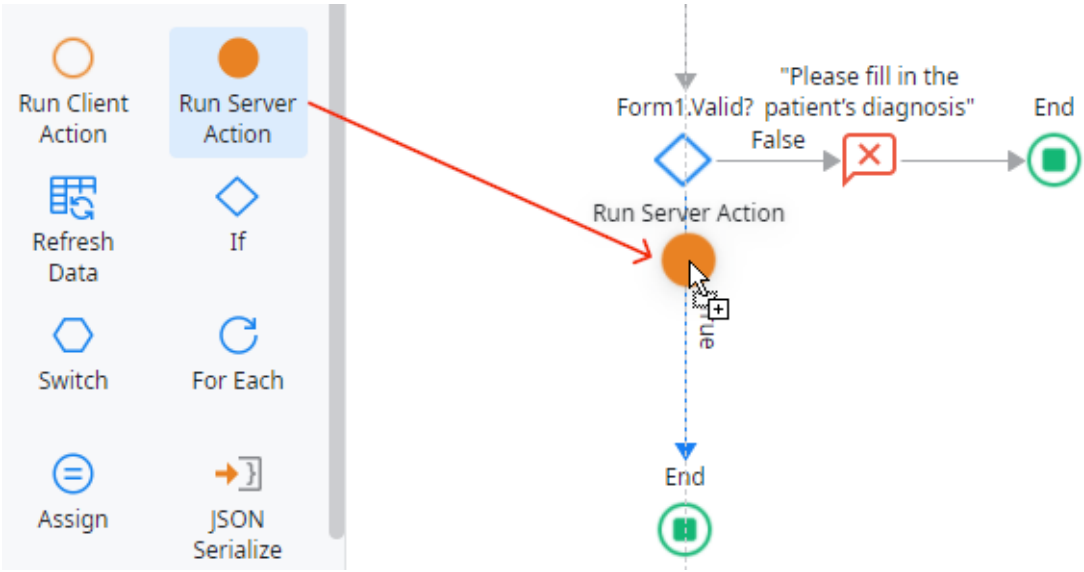
Finish up

The server-side Actions that save the changes in the Appointment and Patient Entities in the Database are ready. You just need to use this logic in the OnClick_Save Action of the Appointment_Detail Screen.

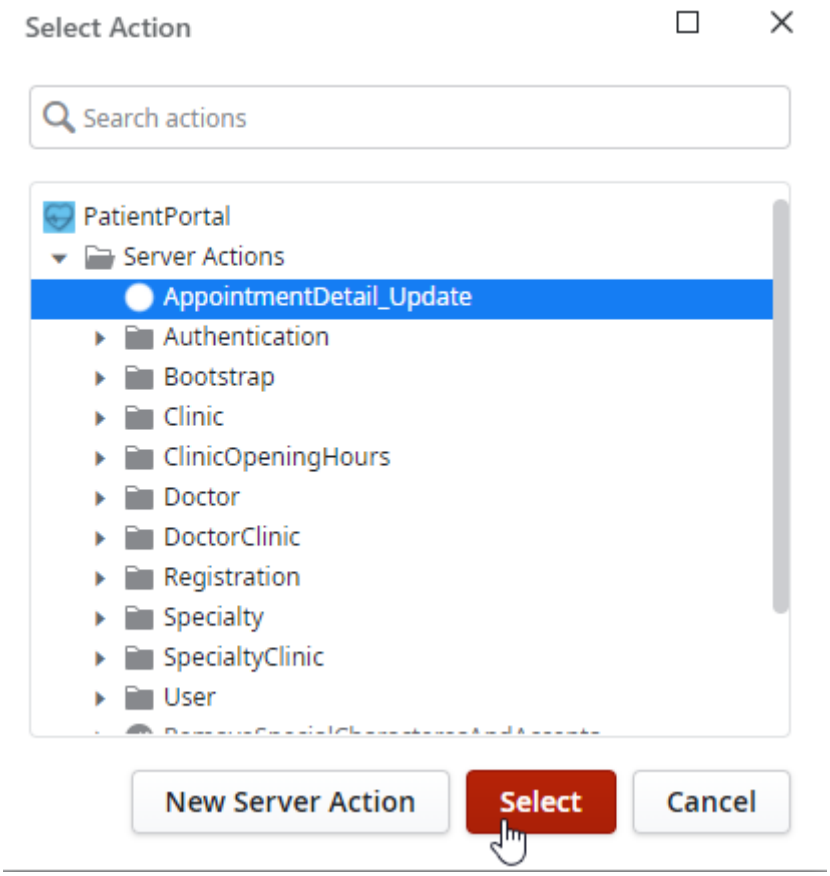
1) Switch to the Interface tab, expand the **Appointment_Detail** Screen and double-click on the **OnClick_Save** Action to open its logic flow.



2) Drag a **Run Server Action** element from the left sidebar and drop it in the True branch.

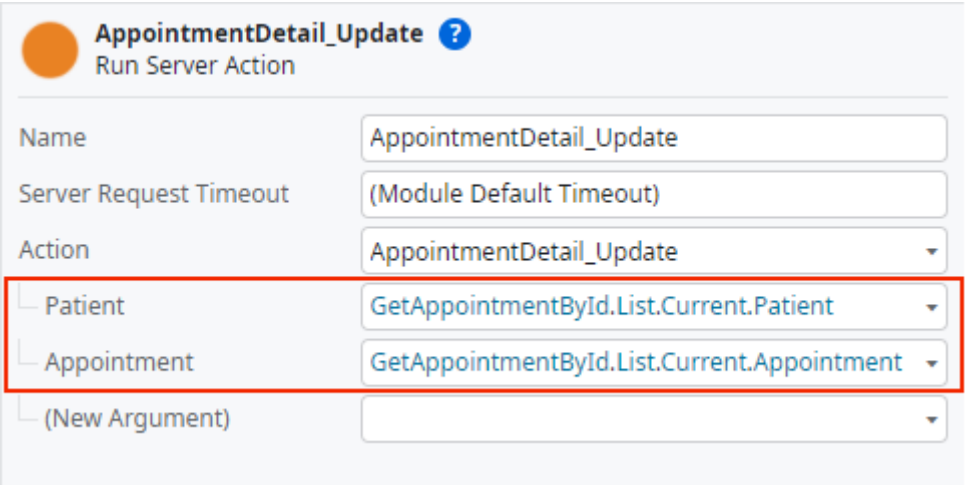


3) Select the **AppointmentDetail_Update** Action in the dialog.

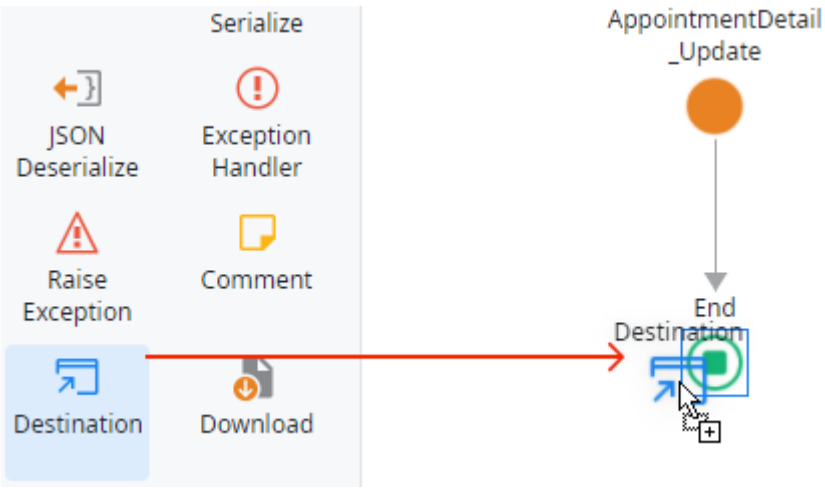


When an Action that has mandatory Input Parameters is called, you need to select the corresponding values to the parameters in the Action that called them.

4) In the AppointmentDetail_Update Action properties, set the **Patient** value to **GetAppointmentById.List.Current.Patient** and the **Appointment** value to **GetAppointmentById.List.Current.Appointment**.

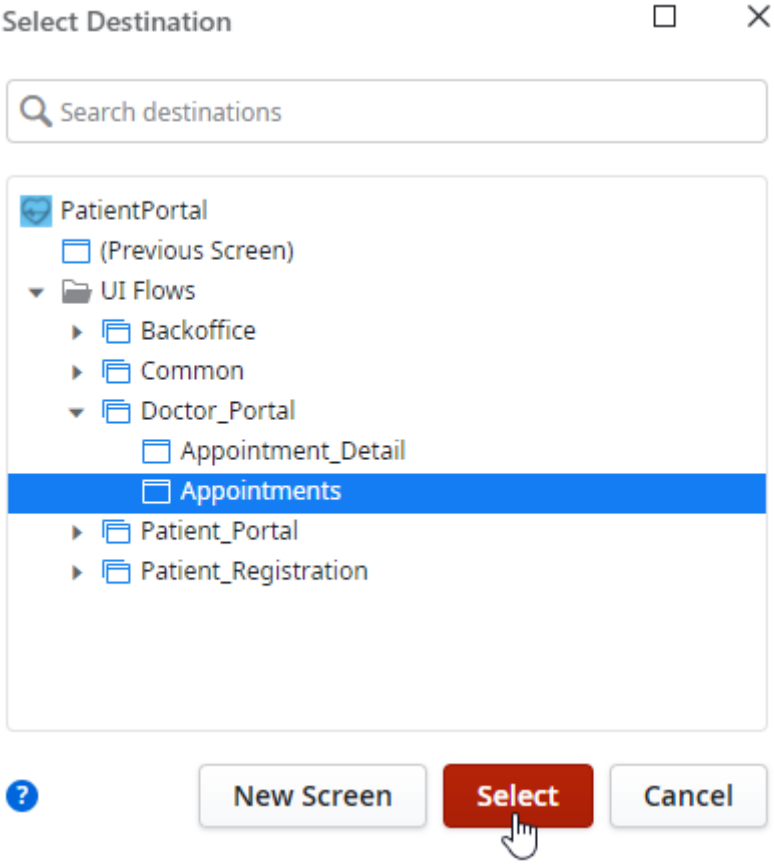


5) Drag and drop a **Destination** element on top of the End node.

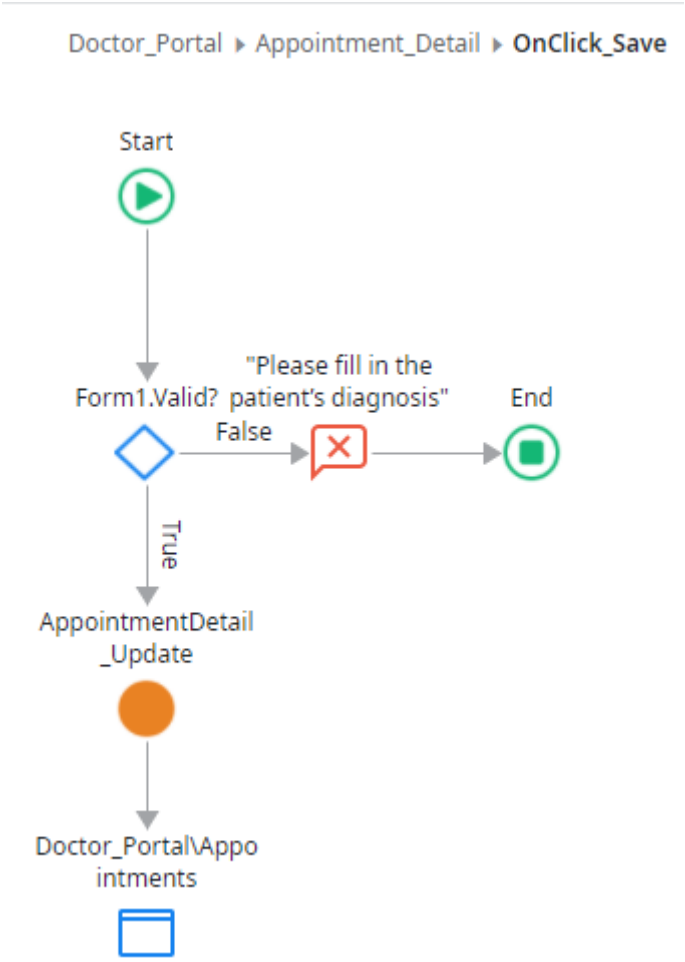


This means the flow will end with the user being redirected to another Screen.

6) Select the **Appointments** Screen in the Doctor Portal flow.



This is what the OnClick_Save Action will look like in the end:



Testing and Results

The Appointment Detail Screen is ready, so it's time to test it!

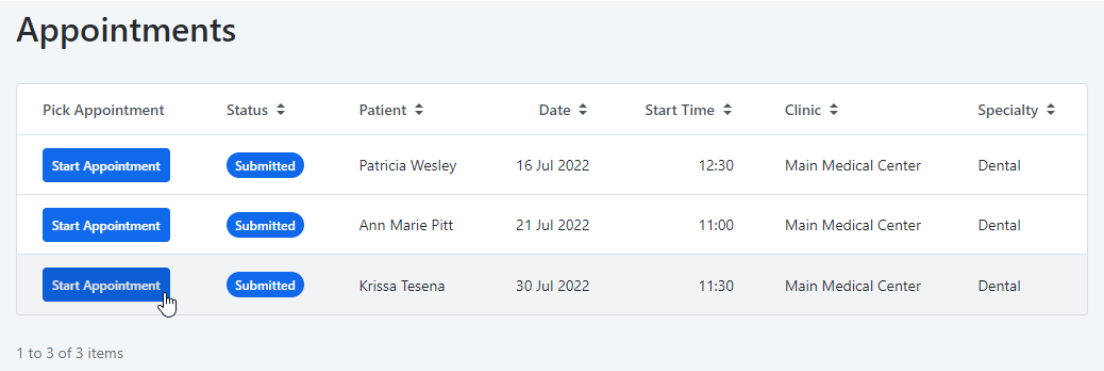
1) Publish the module and open it in the browser.

2) Login as a patient, book an appointment with the doctor Ann Devon, then logout.

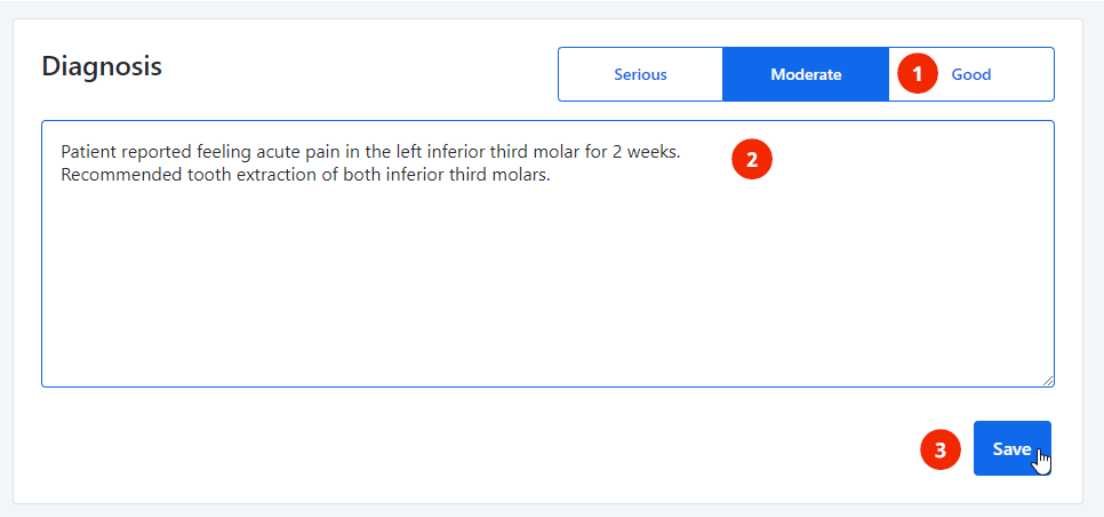
3) Log in as a doctor using the email and password below:

- Username: anndevon@gmail.com
- Password: 1q2w3e4r

4) Click on the **Start Appointment** button.



5) Select the patient health status, type the diagnosis and click on **Save**.



If the operation is successfull, you will be redirected to the Appointments Screen. Otherwise, you should see an error message.

Wrapping up

Congratulations on finishing this tutorial, which is the last one for this lesson. With this exercise, you had the chance to go through some essential aspects of OutSystems, finish up the Appointment Detail Screen, and even get to know more about the platform.

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) Static Entities

2) Server Action

3) Logic

See you in the next tutorial!