



Roles and Input Validations Exercise

 os 

Title*


Title must have a minimum of 3 characters.

Notes

Due Date

Due Date cannot be in the past.

Category

Work 

Priority

LowMediumHigh

Save

Table of Contents

| | |
|--|-----------|
| Table of Contents | 2 |
| Introduction | 3 |
| Built-In Validations | 4 |
| Custom Validations | 9 |
| Role Verification in the Server Action Wrappers | 15 |
| End of Lab | 19 |

Introduction

When developing an application that expects data submitted by the end-users, it is important to safeguard it against potential errors that may occur.

The **ToDoDetail** Screen has a Form with input fields that expect data from the end-users and some validation can and should be done over the data submitted by the user.

First, we will experiment the OutSystems built-in validations, by turning them on and off, using the Service Studio debugger as well.

Then, we will implement two custom validations in the **SaveOnClick** Action, before adding / updating the ToDo to the database. One custom validation will verify if the Due Date of the ToDo is in the past, while the second will verify if the ToDo is being created / updated with a Title smaller than three characters. If one of these validations fail, the ToDo is not sent to the database.

Finally, we will add some verifications to the Wrapper Actions in the ToDo_Core. A ToDo and a Resource can only be created / updated in the database if the user is currently logged in the server. Despite this verification is done at the Screen level, it is best practice to also check the Roles at the server level.

In summary, in this specific exercise lab, we will:

- Learn about the built In validations
- Code custom (business rule) validations

Built-In Validations

When using a Form, with a Link or Button in it, OutSystems provides a set of built-In validations that automates and speeds-up the validation of the data entered in the Form's input fields by the end-user. When the options is turned on, OutSystems automatically validates two built-in scenarios: if all the mandatory fields are filled and if all the data inserted by the user matches the fields data types (e.g: if the user didn't insert a text in a field that expects a number).

In this part of the exercise, we will test and check how the Built-in validations work, and we will use the Service Studio Debugger to help us in that task.

- 1) Try to add a new ToDo to the database, leaving the **Title** empty. Make sure that the operation is not successful.
 - a) Click the Open in Browser button to open the application and select the *Create New To Do* option.
 - b) Fill in the Form leaving the **Title** empty and then click **Save**.
 - c) Notice that an error message appears indicating that a mandatory field is not filled in.

☰ Create New To Do ★

Title*

Required field!

Notes

Due Date

12/17/2018

Category

Work ▼

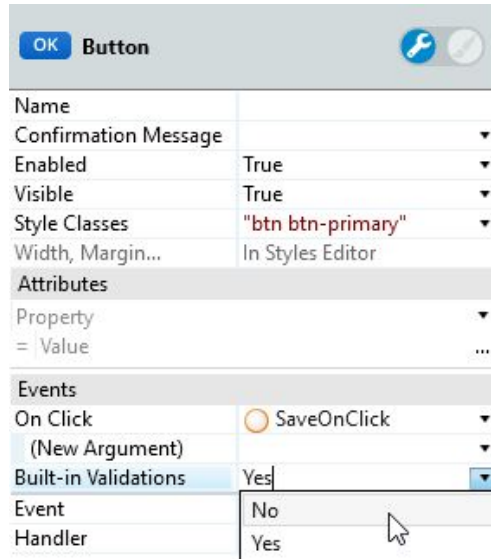
Priority

Low Medium High

Save

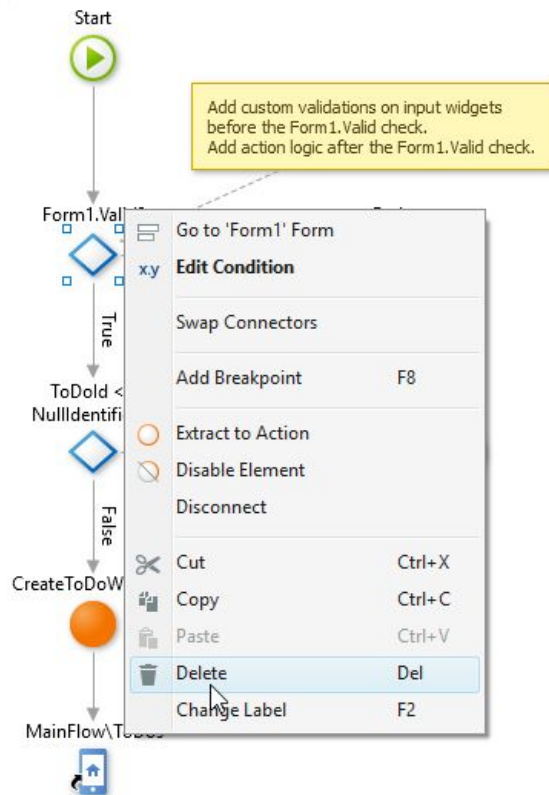
2) Turn-off the built-in validations in the ToDoDetail's Form and do the same test. Notice the differences, since the ToDo is saved in the database.

- a) On the Interface Tab, double-click the **ToDoDetail** Screen to open it.
- b) Select the **Save** Button and set the **Built-In Validations** property value to *No*.



NOTE: This change will allow the data to be sent to the database without any validation, increasing the possibility of errors or incomplete information

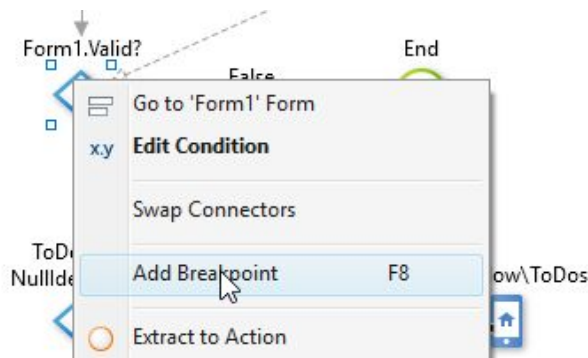
- c) Click the **1-Click Publish** button to publish the module to the server.
 - d) Click the Open in Browser button to open the application and select the option to *Create New To Do*.
 - e) Fill in the Form leaving the Title empty and then click Save. Notice that no errors in submitting the information are reported.
- 3) Turn the Built-in validations property back to *Yes* and, in the **SaveOnClick** Action, delete the verification of the valid property of the Form. Publish and test again to see what happens.
- a) On the Interface Tab, double-click the **ToDoDetail** Screen to open it.
 - b) Select the **Save** Button and set the Built-In Validations property value to *Yes*.
 - c) Open the **SaveOnClick** Action and delete the first **If, Form1.Valid?**



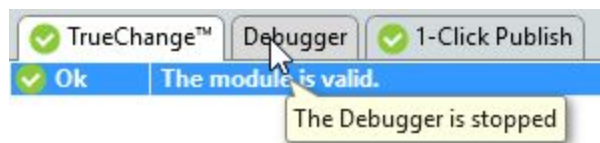
- d) Delete the End statement that is not connected to the flow after the previous deletion.
 - e) Click the **1-Click Publish** button to publish the module to the server.
 - f) Select the **ToDo** that was added in the previous step.
 - g) Leave again the **Title** empty and then click Save.
 - h) Notice that the Save Action was successful again, even if the *Required field!* message appears.
- 4) Revert back the previous changes and put again the If condition in the **SaveOnClick** Action flow. Turn on the debugger and do a step by step execution of the Action, analyzing the Form Valid property in the process.
- a) Revert back the changes by clicking on the Undo option and publish the module.



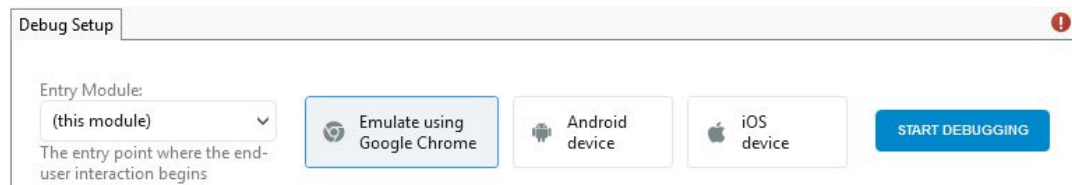
- b) Add a Breakpoint to the *Form1.Valid?* If



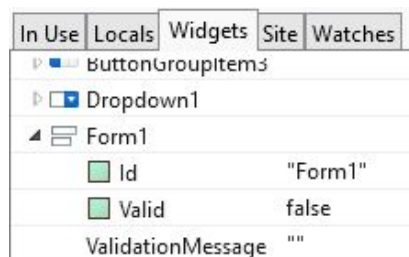
- c) Open the **Debugger** tab



- d) Select the *Emulate using Google Chrome* and click on **Start Debugging**



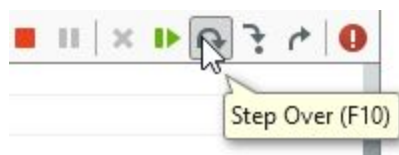
- e) Edit the *ToDo* created above and click on the **Save** Button, still with the **Title** input empty.
- f) Notice that the execution of the application stopped in the browser, and Service Studio appears highlighted. This means that the execution is stopped in the breakpoint.
- g) In Service Studio, select the **Widgets** tab, expand the **Form1** widget and check the *Valid* property of the Form. Make sure it is set to *False*.



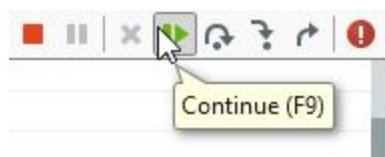
NOTE: Since the Built-in validations are set to *Yes*, and the user did not entered any value for the Title, the **Valid** property of the Form is set automatically to *False*. This is done automatically for the Built-in validations. The Valid property cannot be manually set to False, only automatically when an Input field inside the Form is not valid. As a matter of fact, it takes one invalid Input for the Form to become invalid as well.

Despite the automatic validation, it is important to verify in the Screen Action if the Form is **Valid**, before the data is saved in the database. Without the If, the validations are made but the flow proceeds as designed and bad data can be saved in the database. Service Studio automatically adds the If, when a Screen Action associated with a Button / Link inside a Form is created.

- h) Click on the **Step Over** option and verify that the execution moves to the End statement on the right.



- i) Click on the Continue option to resume the execution.



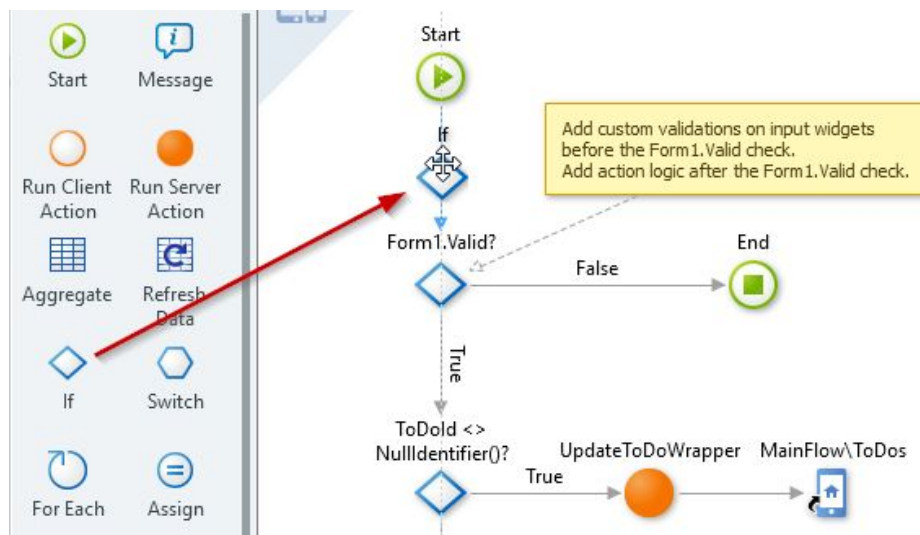
- j) Verify that the error message appeared and the ToDo was not updated in the database.

Custom Validations

Now that we have experimented the Built-in validations, it is time to create some custom validations to the application.

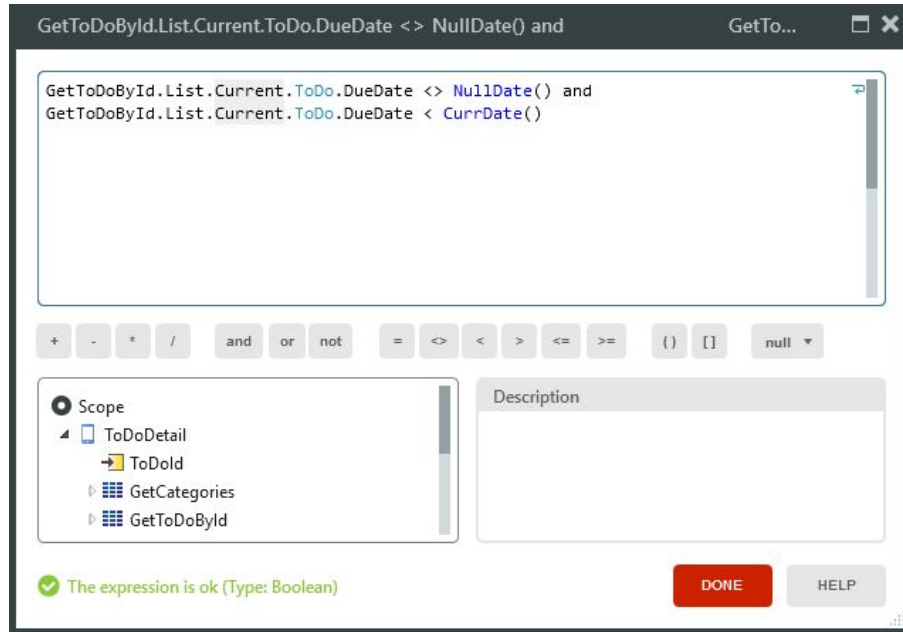
First, an end-user should not save any ToDo in the database with the due date in the past. Then, the Title of the ToDos must be at least three characters long.

- 1) Ensure that a ToDo cannot be saved in the database with a Due Date in the past.
 - a) Open the **SaveOnClick** Client Action of the **ToDoDetail** Screen.
 - b) Drag an **If** statement and drop it between the **Start** and the **Form1.Valid?** statement.

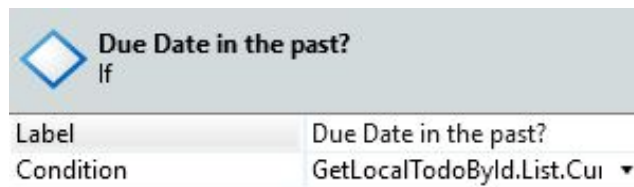


- c) Set the **Condition** property of the created **If** statement to
`GetToDoById.List.Current.ToDo.DueDate <> NullDate()` and
`GetToDoById.List.Current.ToDo.DueDate < CurrDate()`

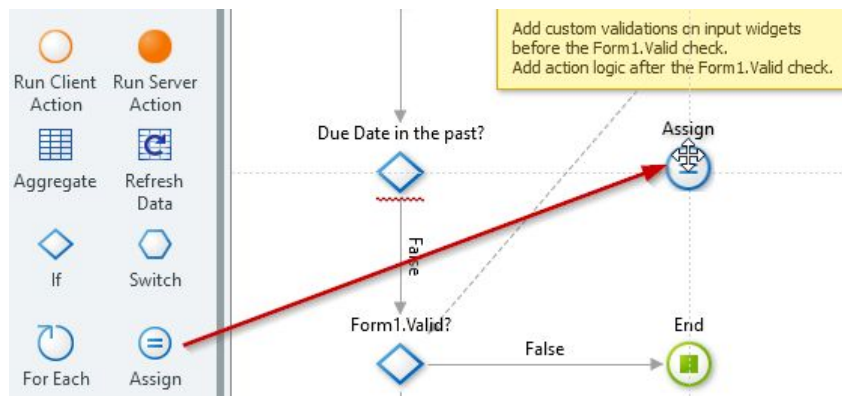
This Condition verifies if the DueDate, when it is not `NullDate()` (input field empty), is prior to the Current Date.



- d) Set the **Label** property of the If to *Due Date in the past?*, to better describe its condition.



- e) Drag an **Assign** statement and drop it to the right of the If statement.



- f) Create the **True** branch connector from the If to the Assign statement.

- g) In the Assign statement, define the following assignments

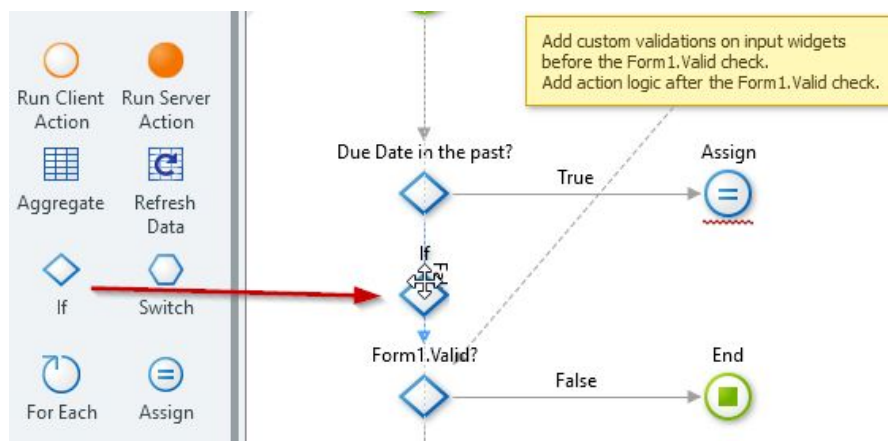
Input_DueDate.Valid = False

Input_DueDate.ValidationMessage = "Due Date cannot be in the past."

| Assign | |
|-------------------------------------|---|
| Label | |
| Assignments | |
| Input_DueDate.Valid | ▼ |
| = False | ▼ |
| Input_DueDate.ValidationMessage | ▼ |
| = "Due Date cannot be in the past." | ▼ |

This Assign statement sets the Valid property of the **Due Date** to *False*, since the Date failed the validation of the If above. Then, the **ValidationMessage** is set to a message that we want to be displayed, next to the input, in the Screen.

- 2) Create a custom validation to ensure that a valid Title must have at least a minimum of three characters.
 - a) Drag an **If** statement and drop it between the **Due Date in the past?** and the **Form1.Valid?** statement.



- b) Set the **Condition** property of the created **If** statement to

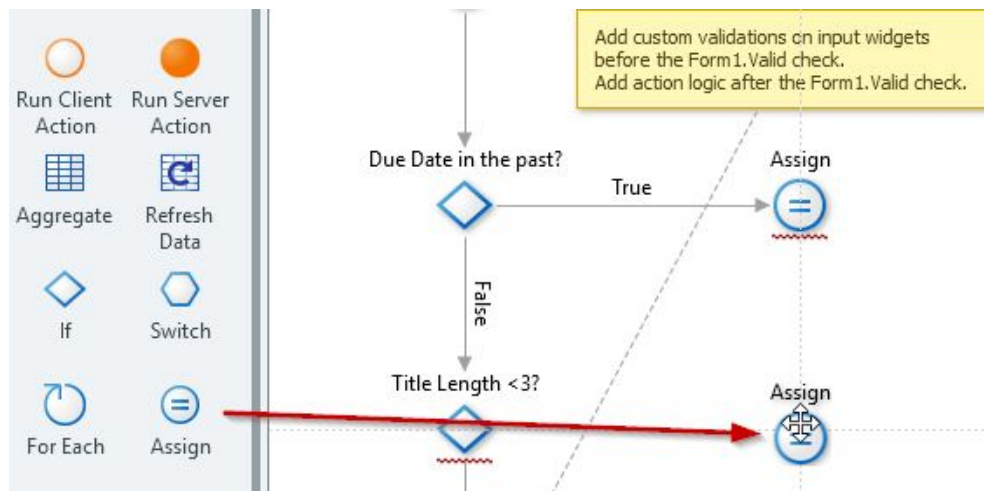
Length(GetToDoById.List.Current.ToDo.Title)<3

This Condition verifies if the Title of the ToDo entered by the user has a length smaller than 3 characters, which would cause the validation to fail.

- c) Set the **Label** property of the If to *Title Length <3?*, to better describe its condition.

| Title Length <3? If | |
|------------------------|-------------------------|
| Label | Title Length <3? |
| Condition | Length(GetLocalToDoBy ▼ |

- d) Drag an **Assign** statement and drop it to the right of the If statement.



- e) Create the **True** branch connector from the If to the Assign statement.

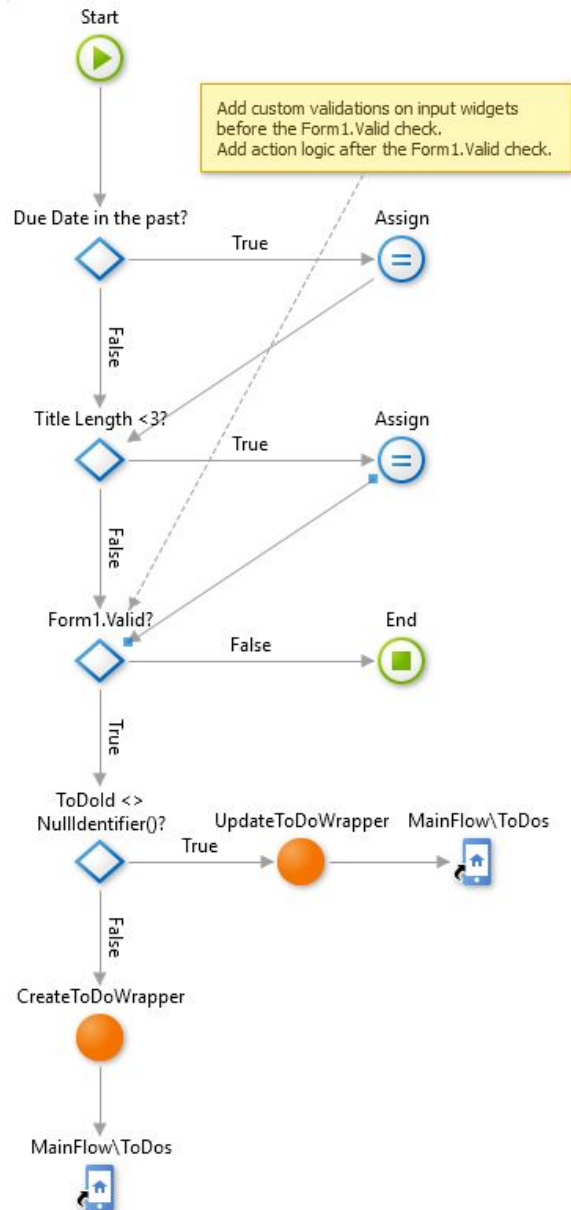
- f) In the Assign statement, define the following assignments

Input_Title.Valid = False

Input_Title.ValidationMessage = "Title must have a minimum of 3 characters."

| Assign | |
|--|---|
| Label | |
| Assignments | |
| Input_Title.Valid | ▼ |
| = False | ▼ |
| Input_Title.ValidationMessage | ▼ |
| = "Title must have a minimum of 3 characters." | ▼ |

- g) Create the connector from the first **Assign** to the **Title Length < 3?** If statement.
- h) Create the connector from the second **Assign** to the **Form1.Valid?** If statement.
- i) The **SaveOnClick** Client Action should look like this.



- 3) Publish the module and test the application. Make sure that it is not possible to save a To Do in the database, with a **Due Date** in the past and the **Title** with less than three characters. Confirm that the validation messages appear as expected.
 - a) Click the **1-Click Publish** button to publish the module to the server.
 - b) Click the Open in Browser button to open the application and navigate to the *Create New To Do Screen*.
 - c) Fill the **Title** with only one or two characters.
 - d) Select a **Due Date** in the past and click on the **Save** button.

os

Title*

os

Title must have a minimum of 3 characters.

Notes

meeting

Due Date

12/12/2018

Due Date cannot be in the past.

Category

Work

Priority

Low

Medium

High

Save

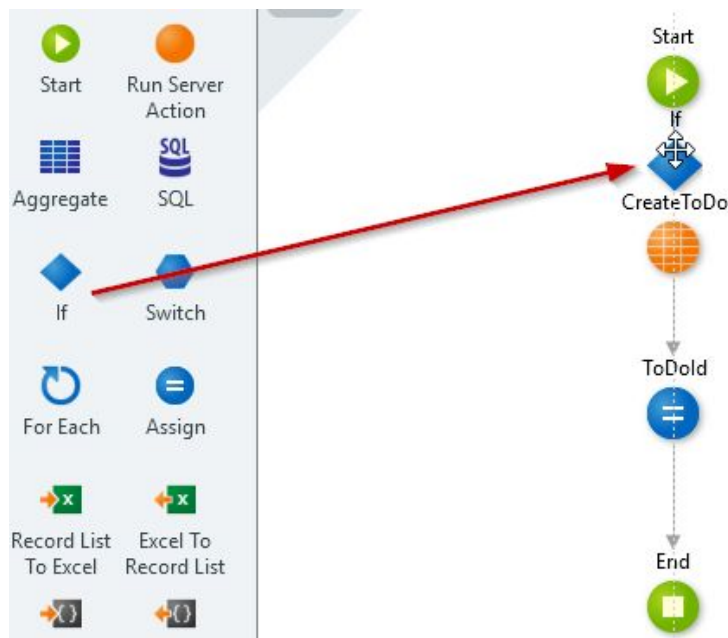
- e) Fix both errors and click **Save** again.
- f) The new To Do should now appear in the **ToDos** Screen.

| Book Hotel |
|---|
| Watch the new Marvel movie with Brie Larson |
| Meeting with manager |

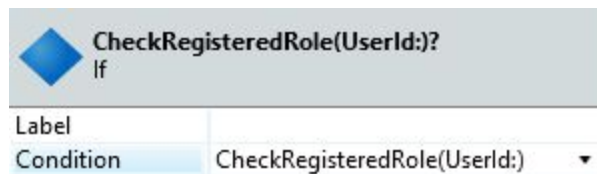
Role Verification in the Server Action Wrappers

In this section of the lab, we will add some Role verification to the Wrapper Actions created in the previous lab. Before creating / updating a ToDo or a Resource, the logic should verify if the user is logged in, meaning if it has the Registered Role. If not, then it should throw an Exception.

- 1) Add a verification to the **CreateToDoWrapper** to check if a user has the **Registered** Role, meaning that has a username and password in the server, before adding the ToDo to the database. Throw an Exception if that is not the case.
 - a) Switch to the ToDo_Core module and open the **CreateToDoWrapper** Action.
 - b) Drag an If statement before the **CreateToDo** Entity Action.

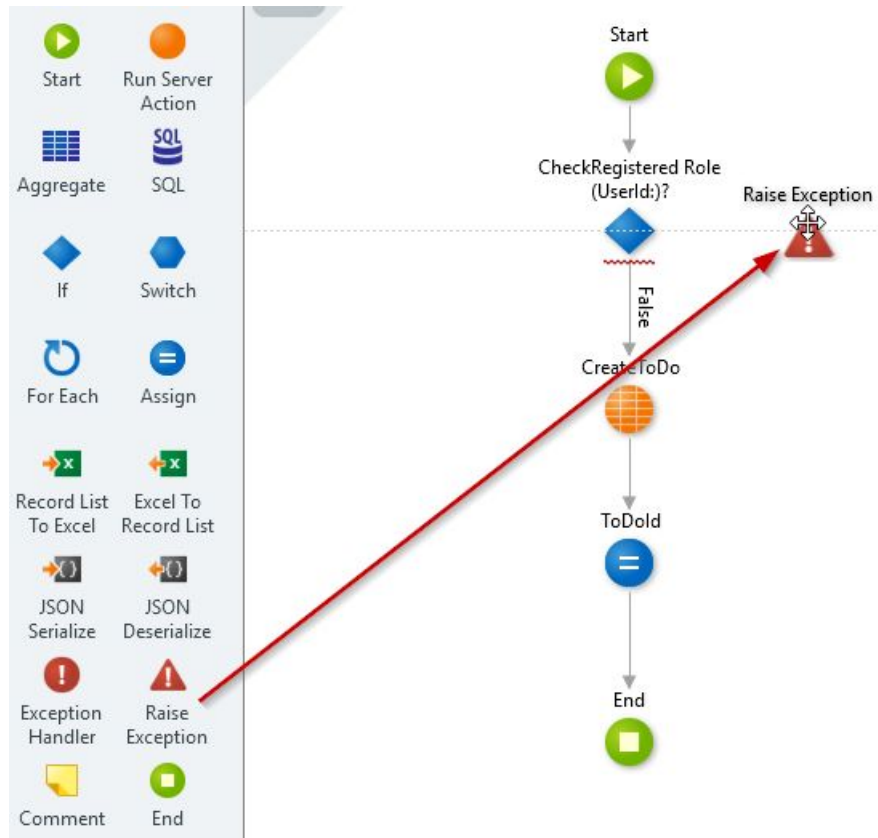


- c) Define the **Condition** of the If as *CheckRegisteredRole(UserId:)*

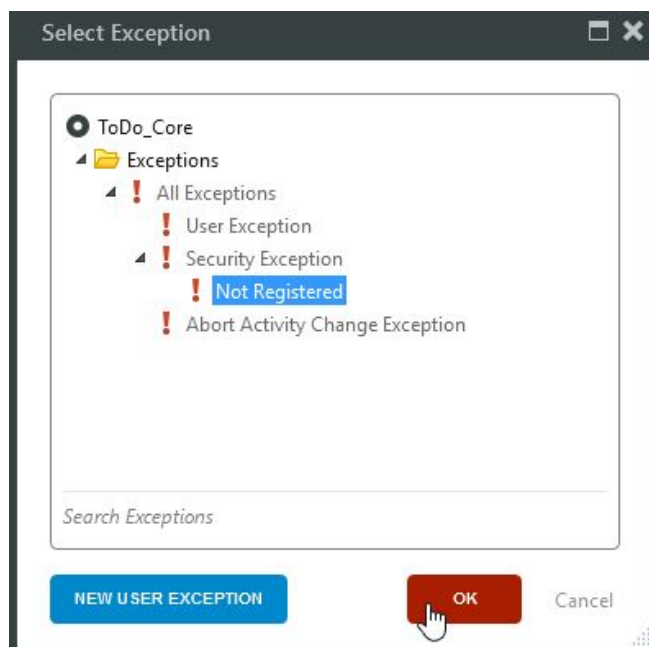


This **Condition** verifies if the user currently logged in (since no parameter is used in the Action) has the Registered Role.

- d) Drag a Raise Exception statement to the right of the If condition.

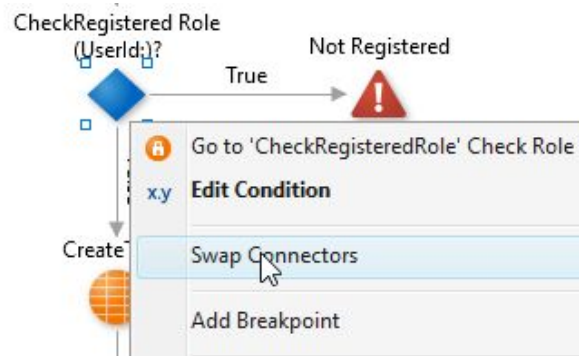


e) Select the **Not Registered** Exception in the next dialog.



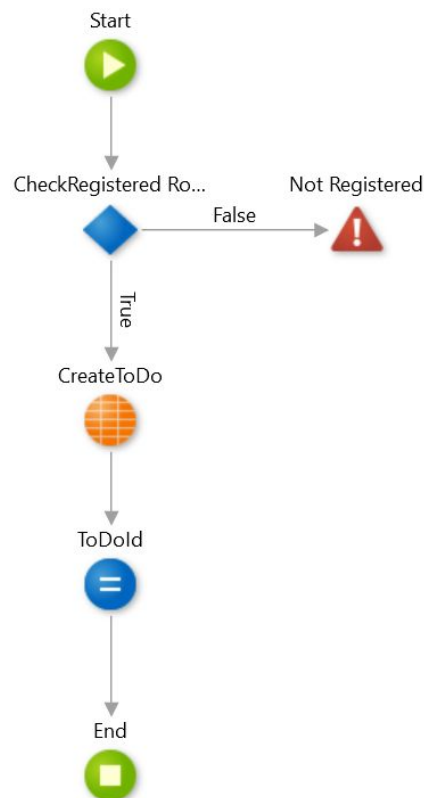
f) Set the **Exception Message** to: *"User must be logged in"*

- g) Connect the If to the Raise Exception statement. Notice that it creates the True branch, which means that the connectors are not correct. Right-click over the If statement and select **Swap Connectors**.



NOTE: The Swap Connectors option can always be used to easily swap the True for the False branch and vice-versa, without needing to redesign the flow of the Action.

- h) The CreateToDoWrapper should look like this



NOTE: When this Exception is triggered in this wrapper, the execution will bubble-up and be handled in the Global Exception Handler. By default, the Security Exceptions (which includes this one) are handled by redirecting the user to the Invalid Permissions Screen, if logged in, or to the Login Screen, if not logged in.

All Screens of the ToDo application are only accessible by Registered users. However, those verifications are done in the device, so it is important to double-check the Role on server-side, before adding data to the database. This pattern is an OutSystems Best Practice.

- 2) Apply the same verification to the other Wrapper Actions, following the same strategy of the previous step.
- 3) Publish the ToDo_Core module and update the references in the ToDo module.
 - a) Publish the ToDo_Core module with all the wrappers having the Check for the Registered Role.
 - b) Switch to the ToDo_core module and publish the module. Since there were no breaking changes to the Wrapper Actions on the ToDo_Core, there is no need to open the Manage Dependencies to update the references.

End of Lab

In this exercise lab, we implemented some validations to the **SaveOnClick** Action in the **ToDoDetail** Screen.

First, we experimented the OutSystems built-in validations, which checks if the mandatory fields are filled in and if the data submitted matches the data types expected on those inputs. This is automatically done by OutSystems and it is up to the developer to use them or not.

Then, we created two custom validations to determine if the Due Date of the ToDo is in the past or if its Title has a length smaller than 3 characters. If one of these scenarios happen, the ToDo cannot be created / updated in the database.

Finally, we added a verification on the Wrapper Actions in the ToDo_Core to determine if the user is logged in, to actual have the permissions to add / update a ToDo in the database.