

# Form Validations

## Exercise

<b>Outline</b>	<b>2</b>
<b>Hands-on</b>	<b>2</b>
MovieDetail Screen	4
PersonDetail Screen	4

# Outline

In this exercise, we will focus on adding some business logic to our app as we validate the forms we have already created. This will be done in the MovieDetail and PersonDetail Screens to avoid saving invalid information in the database.

At the end of this exercise, we want to guarantee that the following rules are followed:

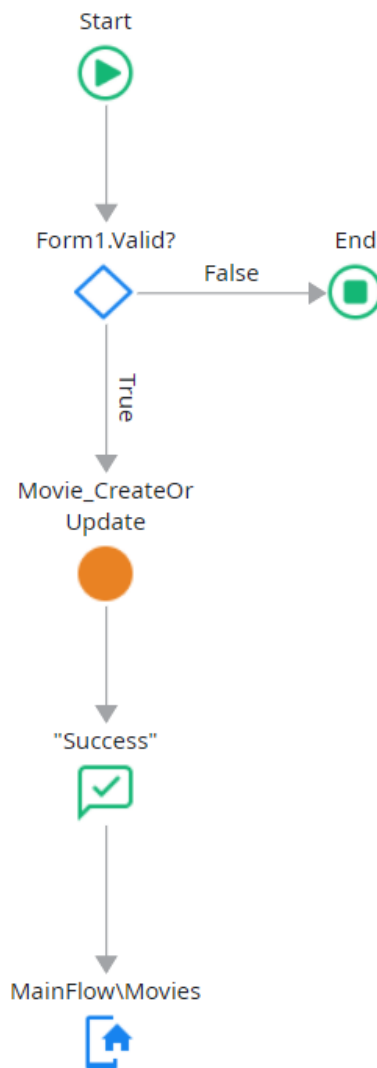
- A movie's gross takings can never be a negative number.
- A movie that wasn't released yet cannot have any gross takings amount.
- A person's date of birth cannot be in the future.
- A person's date of birth cannot be later than their date of death.

If a movie or person was successfully added to the database, then a feedback message should be displayed on the screen.

## Hands-on

In this exercise, we will go back to the MovieDetail and PersonDetail Screens to implement Form validations. As you know, our apps should try to prevent invalid data from being added to the database as much as possible. OutSystems already helps to do so with the built-in validations, but it is up to the developers to guarantee that other business rules are followed as well. With that in mind, we will add some validations to our logic to ensure that our rules are followed.

On both Screens, we already have a Client Action called SaveOnClick which is triggered when the Save button is clicked. Those Actions hold the logic to add a movie/person to the database depending on the Screen we are in. When opening one of those Actions, we can see that the platform automatically adds an If statement to the flow which checks to see if the Form is valid or not.



This guarantees that if the built-in validations (mandatory fields and wrong data types) fail, then the movie or person is not created/updated in the database.

Now, we want to add our custom validations to both Screens.

## MovieDetail Screen

The first thing we want to assure is that movies in the database do not have a negative gross takings amount. If this scenario occurs, then the movie cannot be created/updated in the database and the following message should appear to the end-user:

*The gross takings amount cannot be negative.*

Next, we want to guarantee in the MovieDetail Screen that a movie that hasn't been released yet does not have a gross takings amount. In other words, movies with a release

date in the future cannot have a gross takings amount greater than zero. Just like in the previous scenario, if this happens, the movie cannot be created/updated in the database and the following message should appear to the end-user:

*The movie cannot have any gross takings since it was not yet released.*

**Hint:** To check if a movie has a release date in the future, the built-in function *CurrDate()* can be helpful to find the current date. Also, the *Year()* function returns the year of a given date.

When everything goes well, ensure that the user gets a feedback message indicating success.

## PersonDetail Screen

We also want to add two validations to the PersonDetail Screen.

First, we want to ensure that a person's birth date is not in the future. When a user adds a birth date that does not meet this requirement, the person cannot be created/updated, and the following message should appear:

*The date of birth cannot be in the future.*

Then, we need to guarantee that a person does not have their date of death before their date of birth, meaning that a person cannot have died if they have not been born yet. When a date breaking this validation rule is sent to the server to be saved in the database, the following message should appear to the user:

*The date of death cannot be before the date of birth.*

Just like with movies, when the validations are successful and the person is added to the database, a feedback message indicating success should be displayed to the user.

Don't forget to test the validations with real data to make sure everything works!