

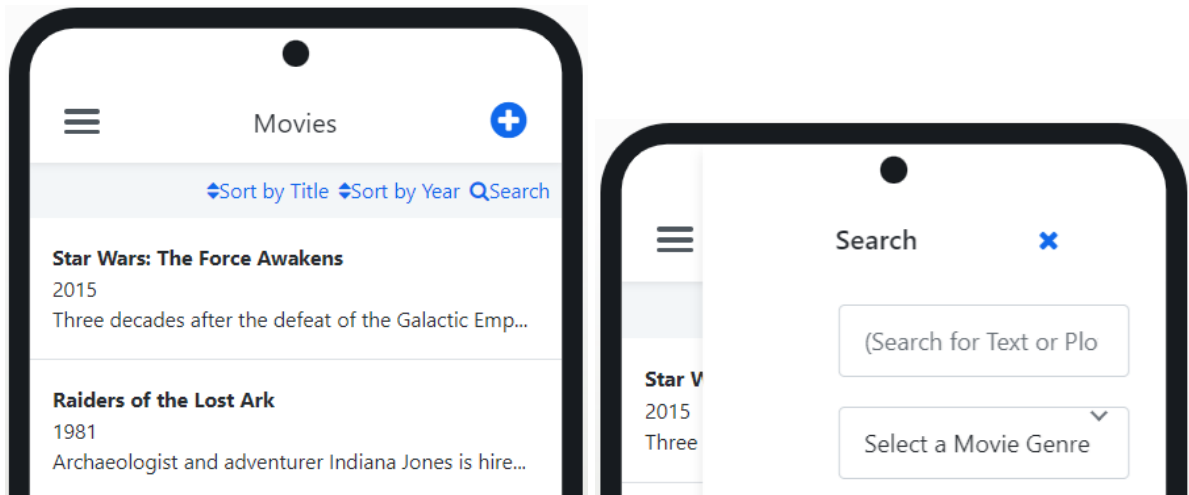
Search Filters

Exercise

Outline	2
How to	2
Adding Search Input Fields	3
Applying the Search	18

Outline

In this exercise, we will expand the functionality of the app with a new filter functionality on the Movies Screen. We want our users to be able to search by movie title, plot summary, and movie genre and we want to make sure that when we change to a different Screen and come back, the search filter is not lost. The search filter should look like this:



Note that the search should be done automatically when the users finished typing the title/plot summary keyword, or when they just chose the movie genre. So, we don't want a button/link to trigger the search. Also, the end-user does not need to type the whole movie title and the whole plot summary, for the search to return the movies that match the keyword that was typed.

This needs to be done in two steps:

1. The UI part, where we need to add input fields to allow searching for the keywords.
2. The logic part, where we need to adjust the Aggregate that fetches the movies, save the search filters in variables that persist even when we change Screens, and trigger the search logic automatically. For this, we will use Client Variables.

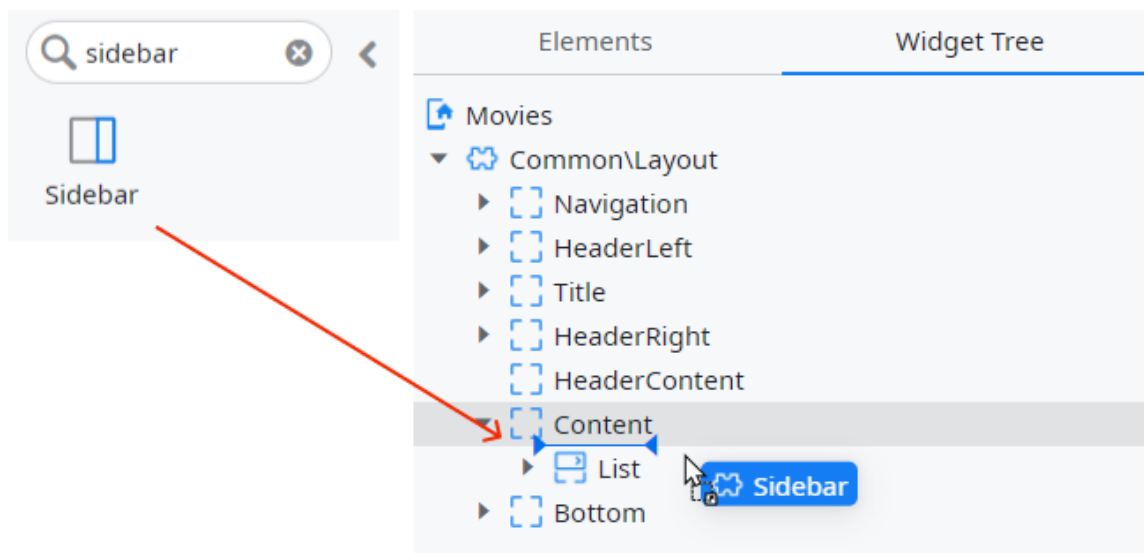
How to

In this section, we'll describe exercise 9 – *Search Filters*, step by step.

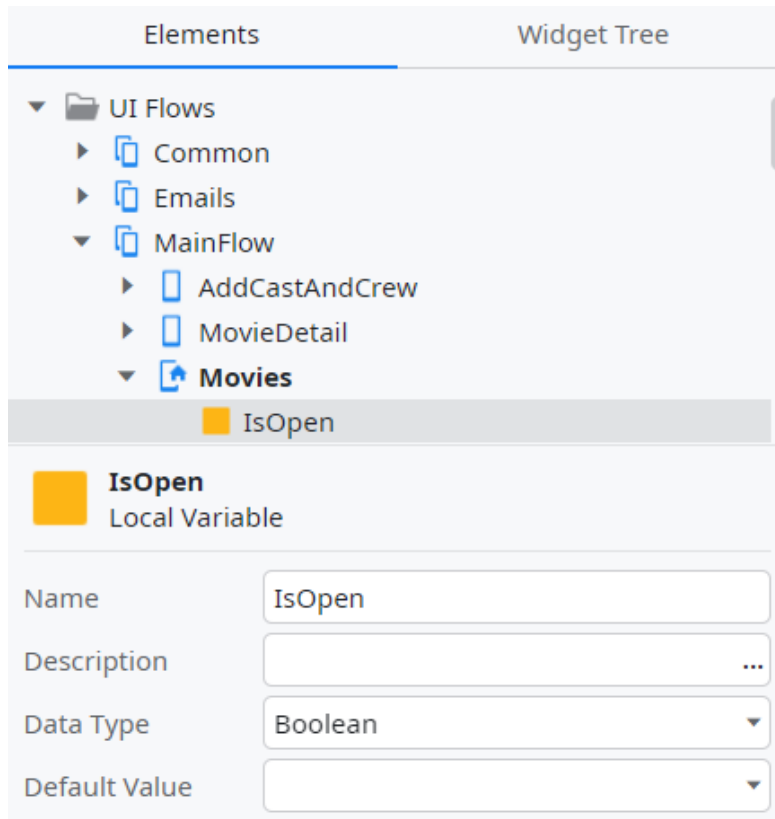
Adding Search Input Fields

To allow searching for movies by title/plot summary and movie genre, we need to add input fields to the Screen where the end-user can define the desired search filters. To avoid overpopulating the Screen, these input widgets will be added to a Sidebar, which can be used whenever the user wants to search for something on the Screen.

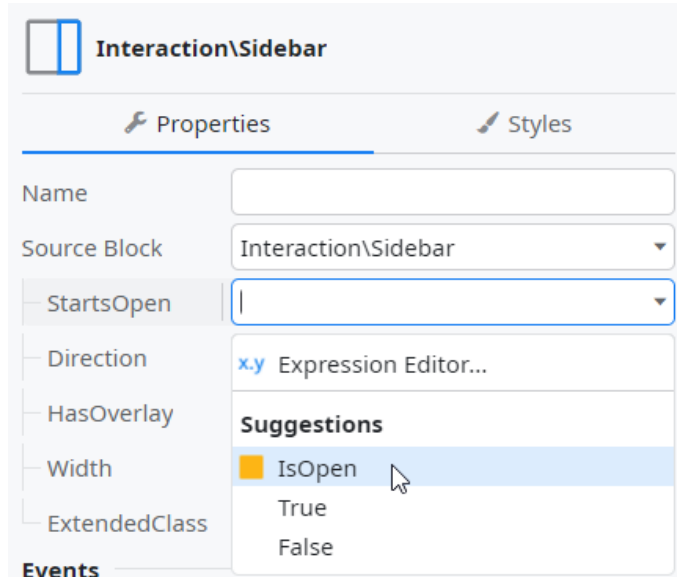
1. Drag a **Sidebar** widget to the top of the Movies Screen, right before the List of Movies, and create a Boolean variable to control when to open and to close the sidebar.
 - a. Start by finding space on the Screen for the Sidebar widget. In the **Movies** Screen, open the Widget Tree, expand the **Content** and drag the **Sidebar** widget above the List widget. Name it *SearchSidebar*.



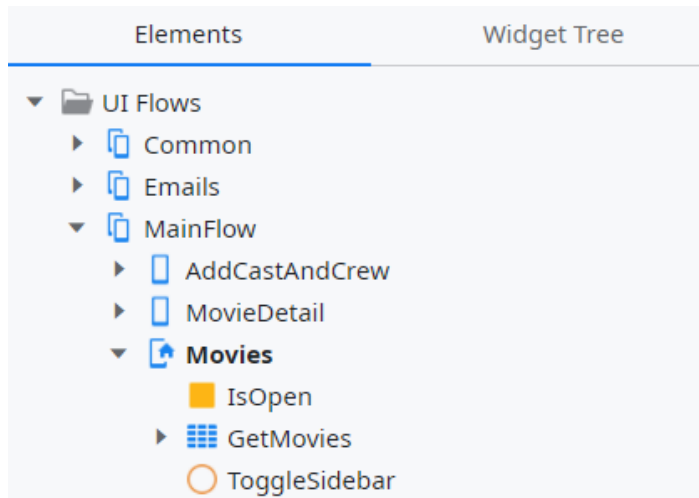
- b. Create a **Local Variable** on the Movies Screen. **Name** it *IsOpen* and make sure its **Data Type** is set to *Boolean*.



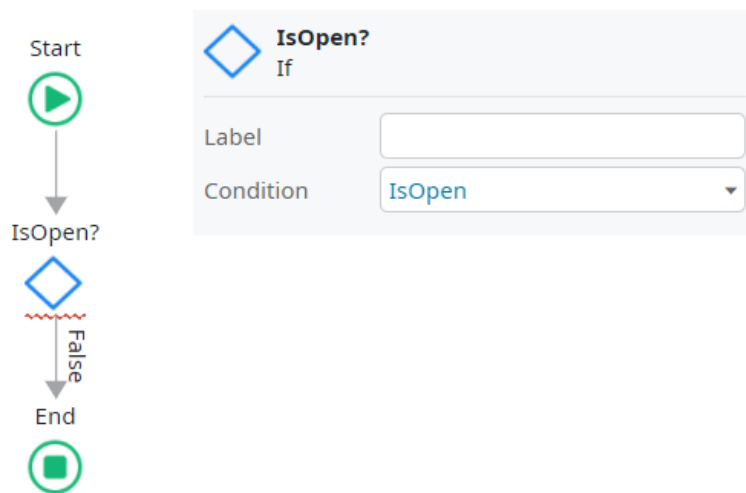
- c. In the **Sidebar** widget, select the new variable created in the attribute *StartsOpen*.



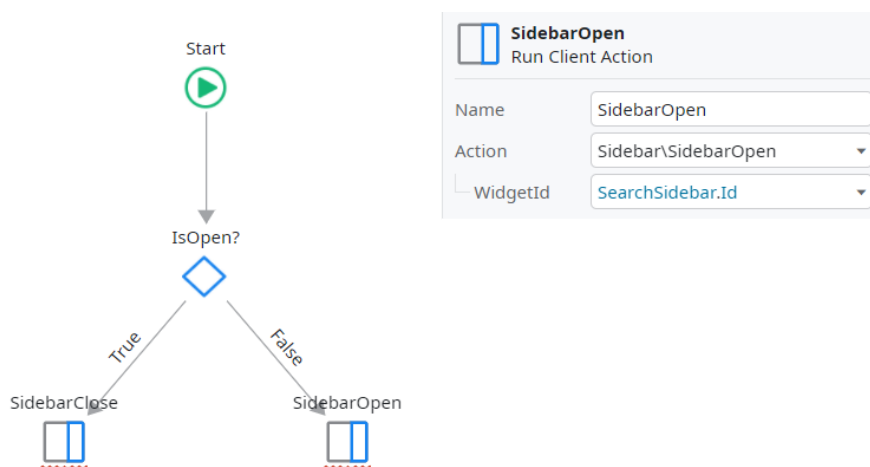
- d. Create a new **Client Action** and name it *ToggleSidebar*.



- e. Add an **Assign** and set its condition to *IsOpen*.



- f. Add a **SideBarClose** widget to the True branch and a **SideBarOpen** to the False branch. Set the **WidgetId** to *SearchSidebar.Id*.



- g. Add and Assign with the following assignment *IsOpen = not IsOpen*

IsOpen

Assign

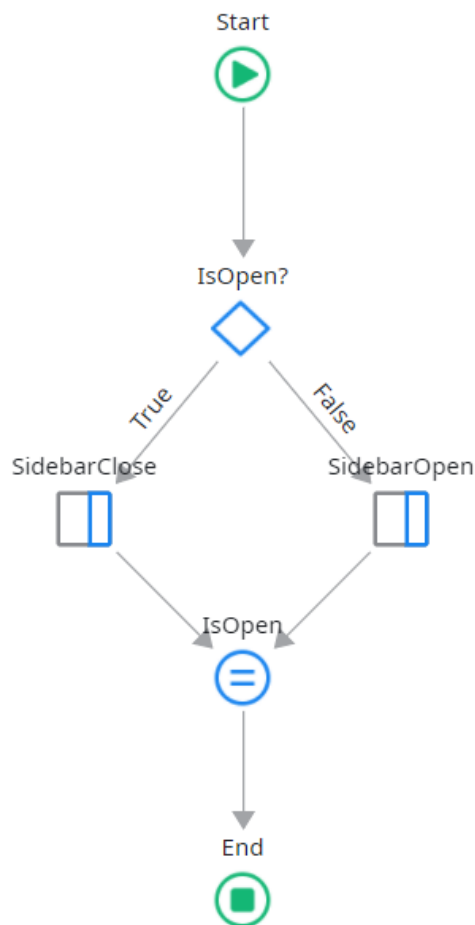
Label

Assignments

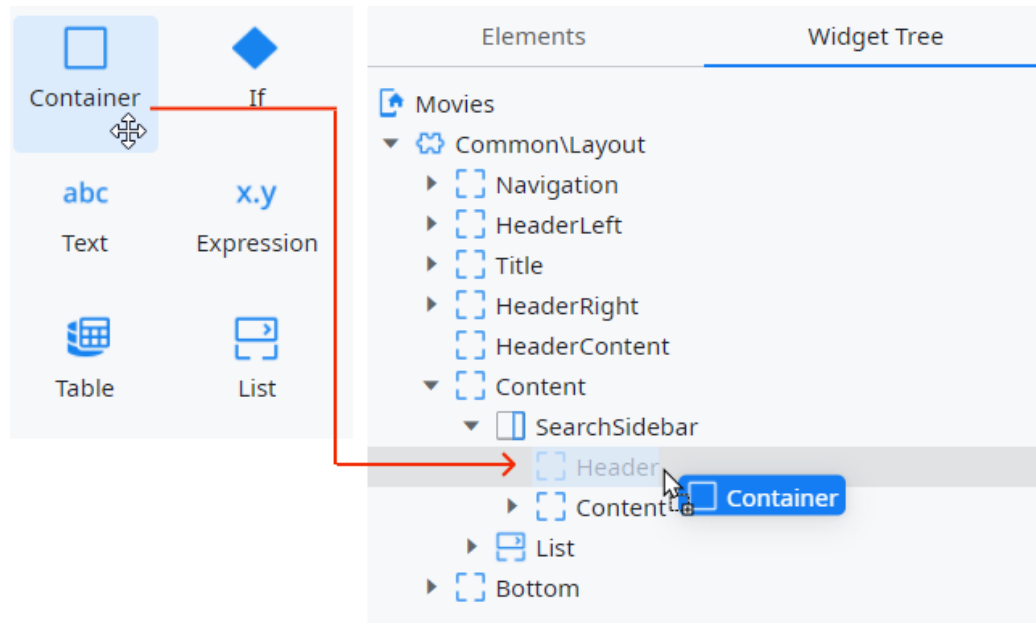
IsOpen

x.y = not IsOpen

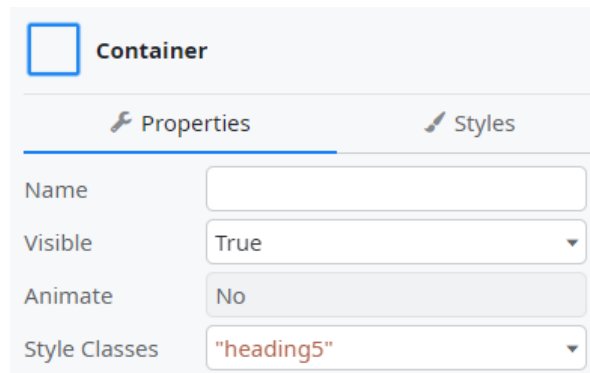
h. Your ToggleSidebar flow should look like the following screenshot:



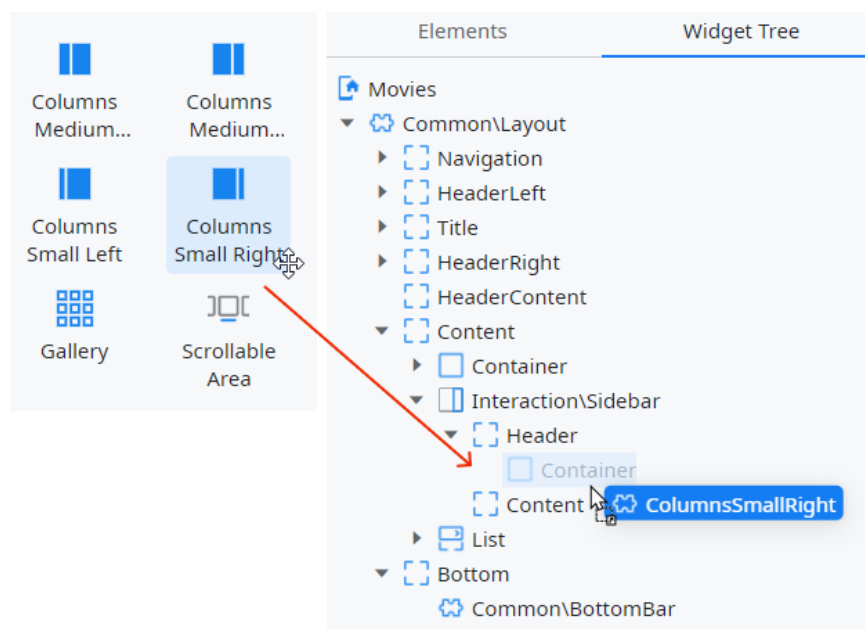
2. Define the header of the Sidebar to have two columns: one bigger on the left, with the text Search in it, and one smaller on the right with an icon (remove icon) to close the Sidebar. These elements should have the "heading5" style applied to it. Use **Containers** and the **Columns Small Right** widget to help.
 - a. Open the Widget Tree, drag and drop a **Container** in the **Sidebar's Header** section.



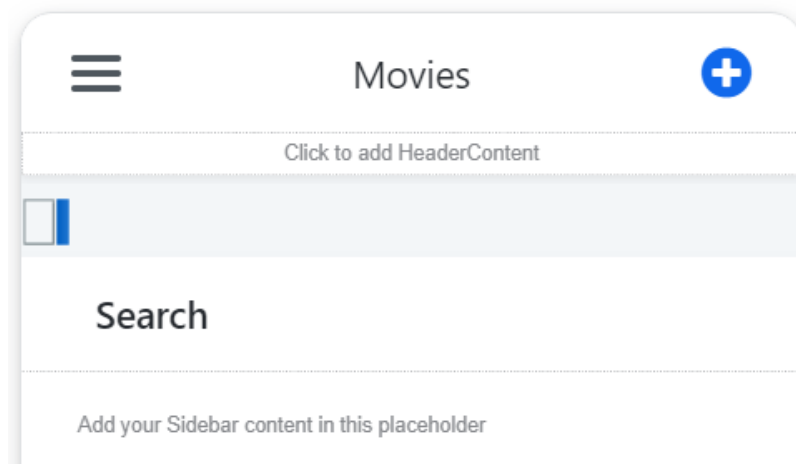
- b. Set the **Style Classes** property of the Container to "heading5".



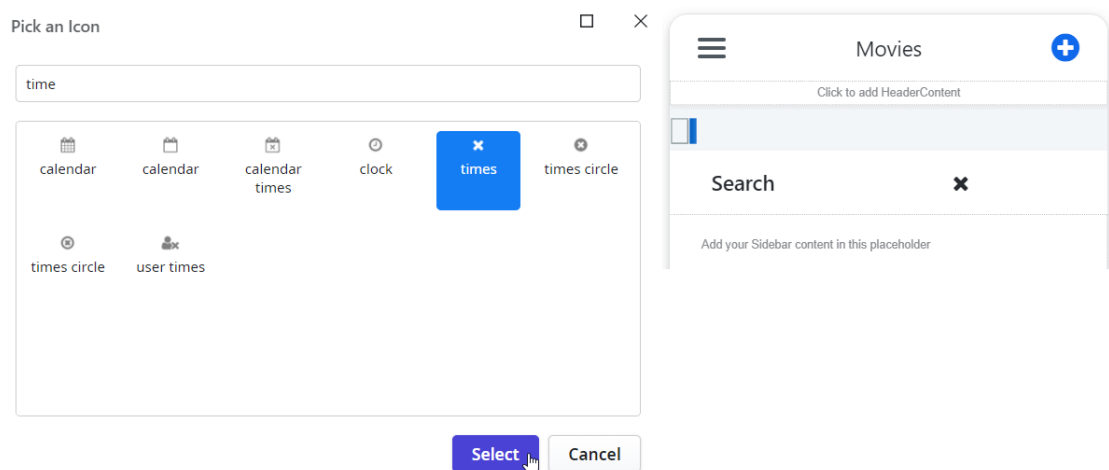
- c. Drag a **Columns Small Right** widget and drop it in the recently added Container.



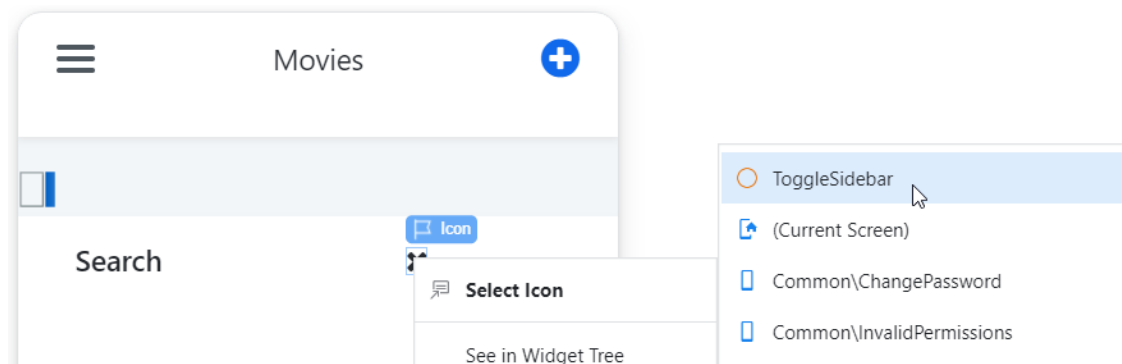
- d. On the **Column1** section of the recently dragged Columns Small Right widget, type *Search*.



- e. Drag an **Icon** widget and drop it on the second column (**Column2**). Choose the *times* icon and set its **Size** to *Font Size*.



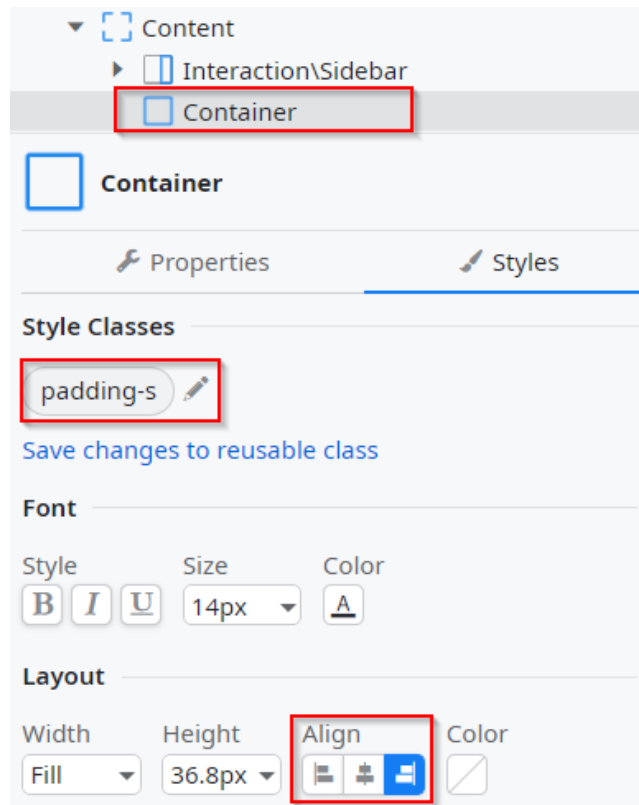
- f. Right click on the **Icon** and **Link** it to *ToggleSidebar* Client Action.



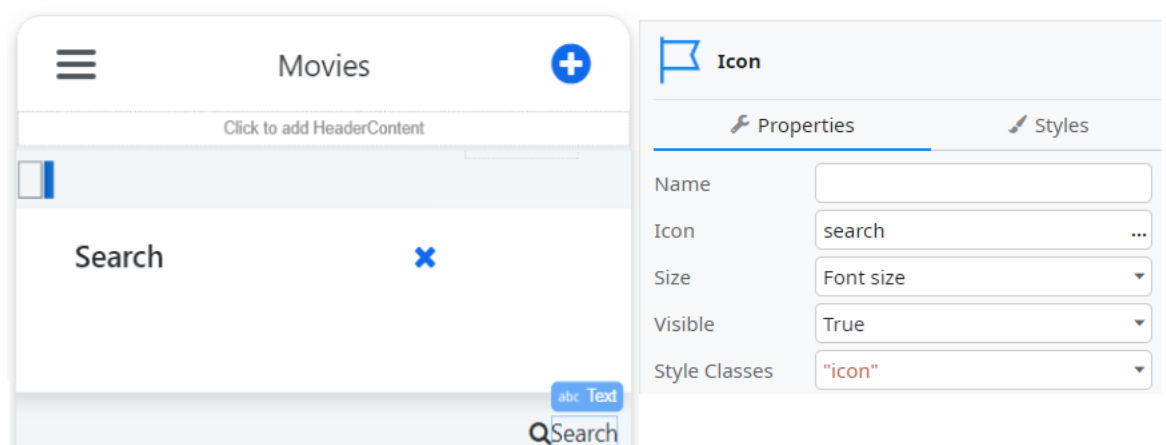
3. On the Movies Screen, a section right above the list of movies that will open the Sidebar. That section should be aligned to the right of the Screen and should include

a *magnifying glass* icon followed by the text Search. When clicked, the Sidebar should open.

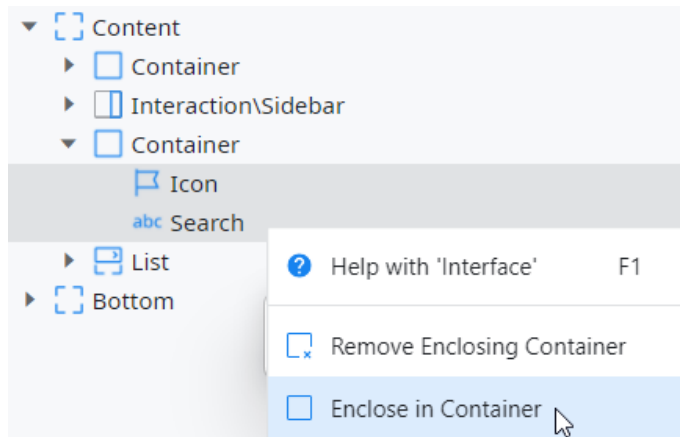
- a. Drag a **Container** and drop it above the List of movies. Change its alignment to the right and add the class *"padding-s"* to add margins around the Container.



- b. Drag an icon and drop it in the Container. Select the search icon and set its **Size** to Font size. Type the word *Search* right next to the icon.

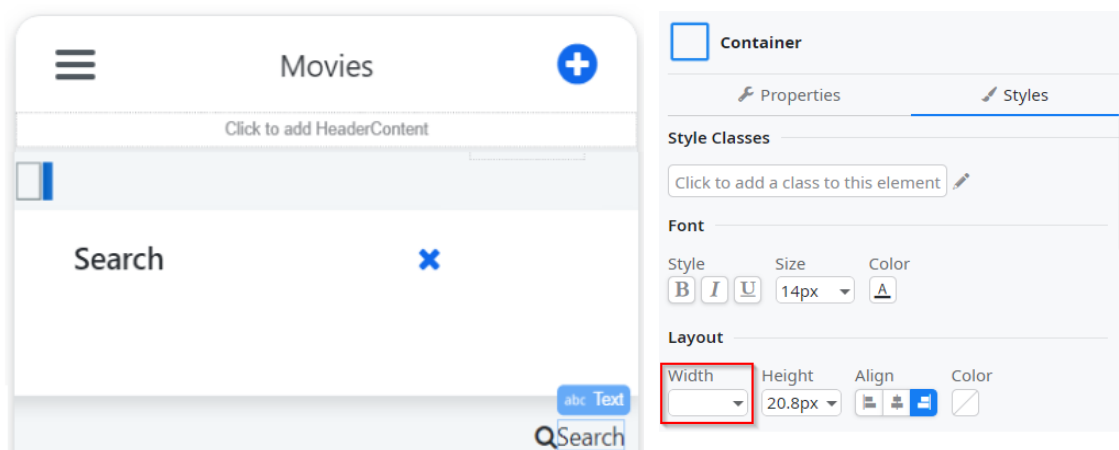


- c. Select both the **Icon** and the **Text**, right-click the selection and choose the option **Enclose in Container**.

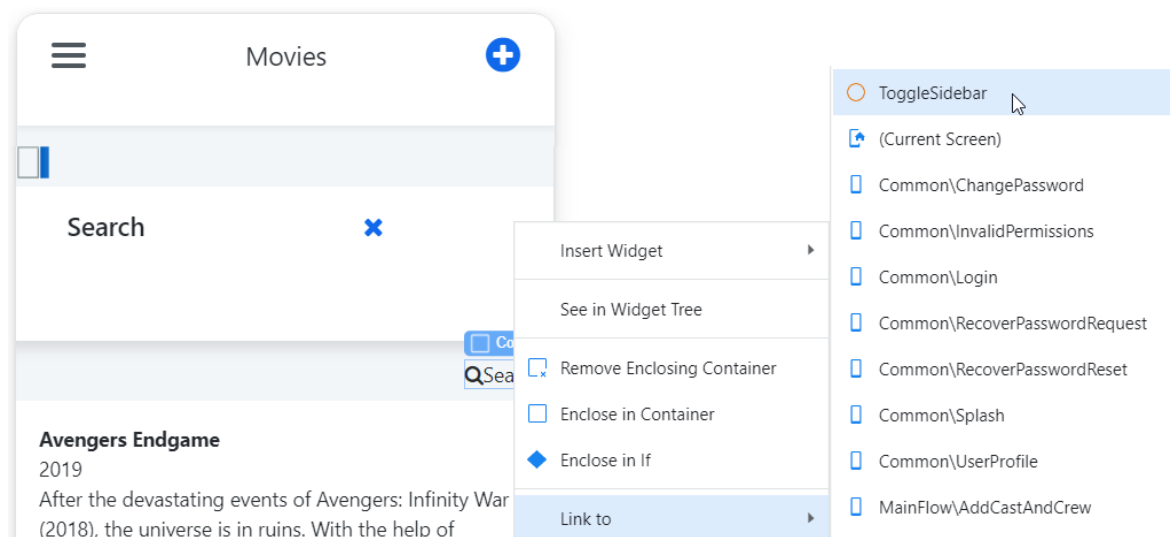


Note: Enclosing two widgets in a Container will automatically set the width of that Container to *Fill*. To make the width match the content we have, we need to change the Width property of the Container and leave it blank.

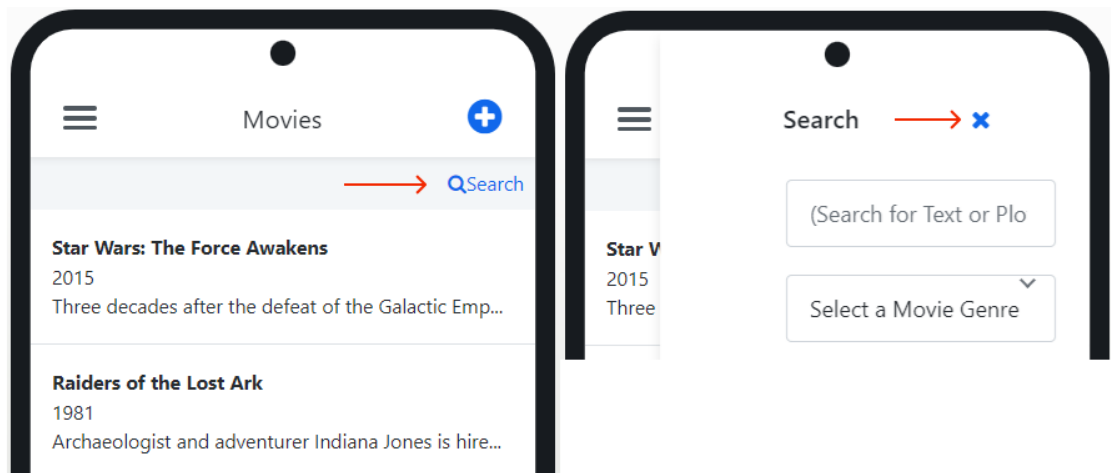
- d. On the recently added Container, open its **Styles** tab and delete the Width field to leave it blank.



- e. Right-click on the Container and **Link** it to the **ToggleSidebar** Client Action.

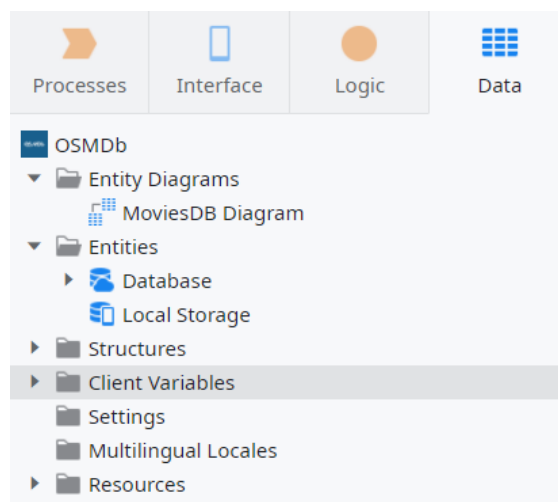


- f. Publish the module and open it in the browser. Make sure the Slidebar opens and closes properly.

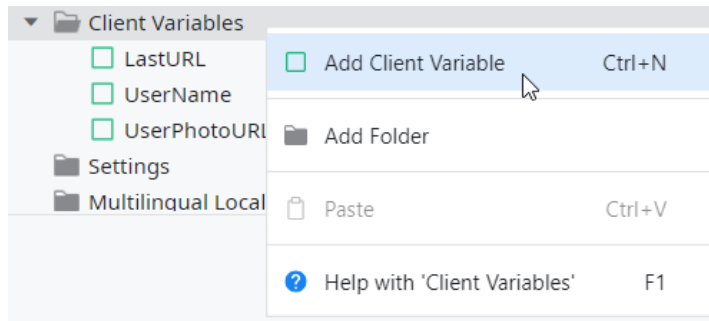


4. Now that we have prepared the Sidebar, it is time to actually create the filter inputs. We will add an input field to search for a textual keyword, that will be applied for the movie title and plot summary, and a dropdown for the movie genre. The values that the user will submit on the search filters will be saved in two Client Variables, so that when the user changes Screen and comes back, the search filter is not lost.

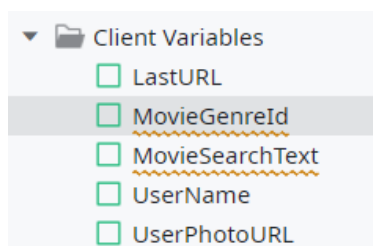
- a. Switch to the Data tab in Service Studio and locate the **Client Variables** folder.



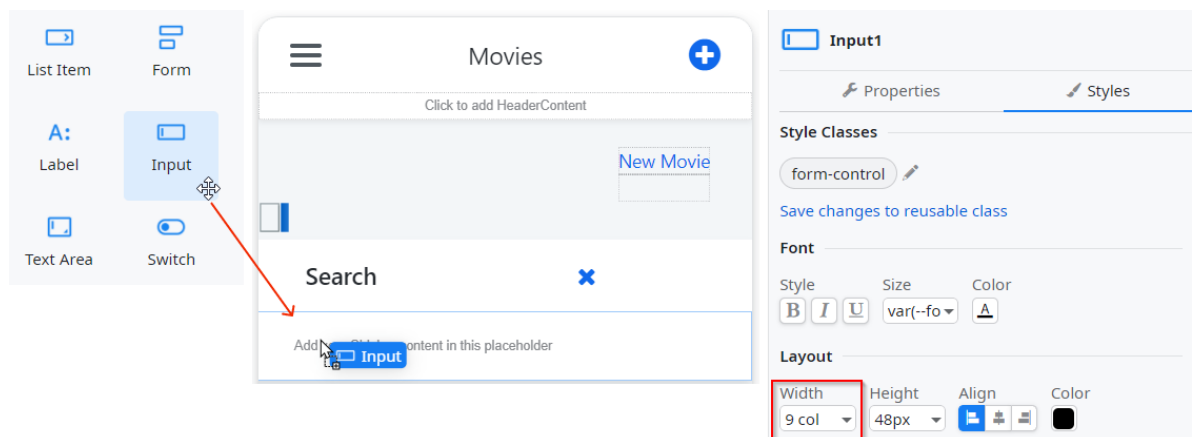
- b. Right-click on the Client Variables folder and select **Add Client Variable**.



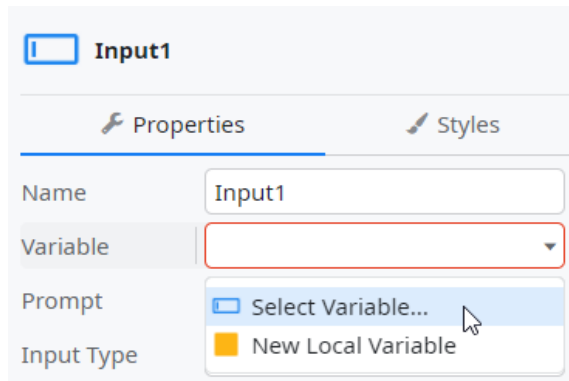
- c. Set the **Name** of the Variable to *MovieSearchText* and make sure the **Data Type** is set to *Text*.
- d. Repeat the previous two steps, but this time for a Client Variable named *MovieGenreId* of **Data Type** *MovieGenre Identifier*.



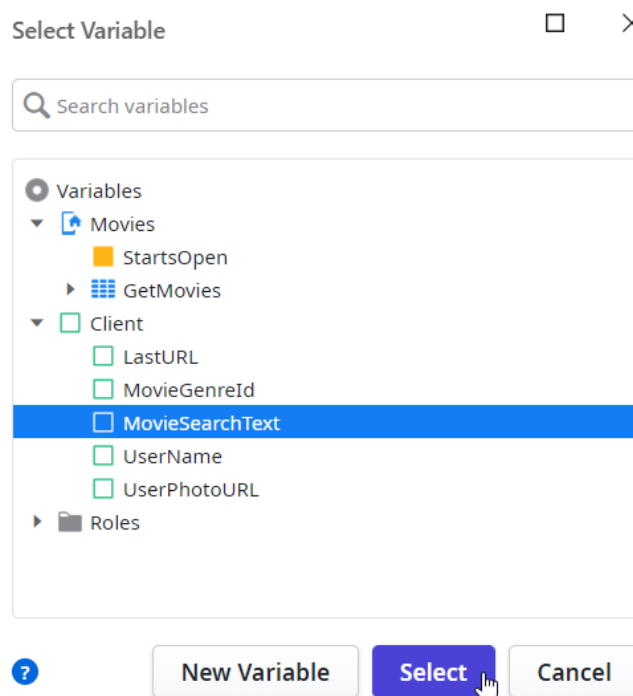
- e. Go back to the Movies Screen, drag a **Input** widget and drop it on the **Content** section of the Sidebar. Define its **width** to 9 col/.



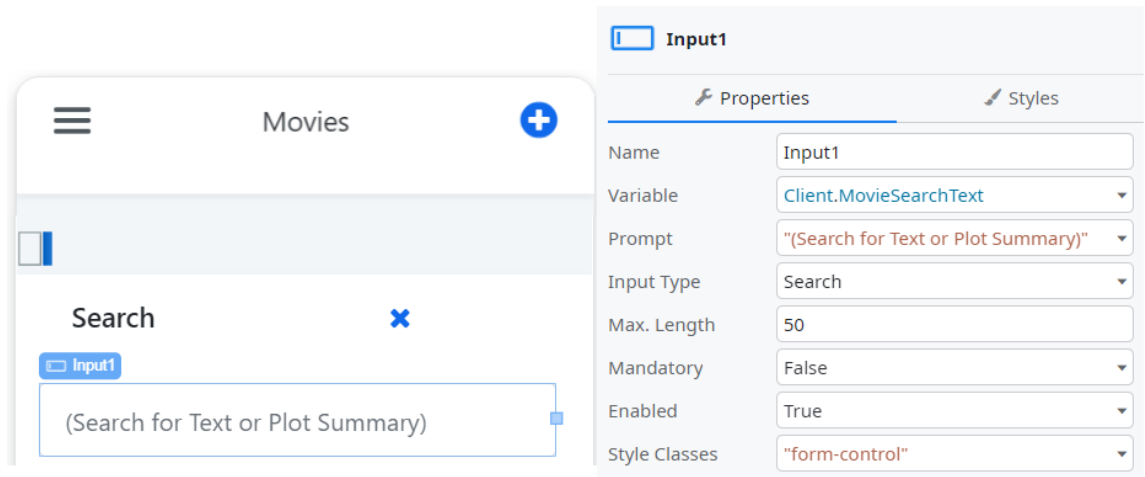
- f. Go to the Input properties area and on the Variable property choose the option (Select Variable...).



- g. In the new dialog, expand the Client Variables section and select the *MovieSearchText*.

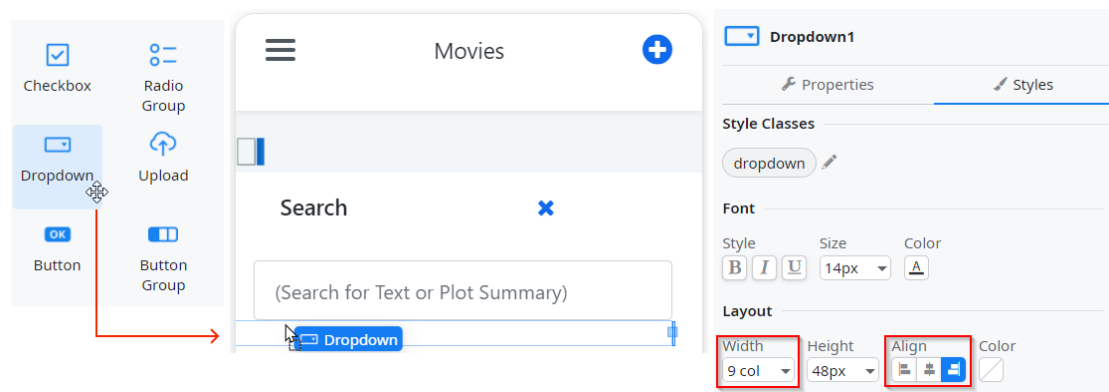


5. Now we need to finish the implementation of the input widgets to remove the errors and also help the end-user understand which search filters are allowed.
- a. Set the **Prompt** property of the Input field to *"(Search for Text or Plot Summary)"*. This message will appear to the end-user, inside the Input field, before the user types any text, which is helpful to explain the purpose of the field.

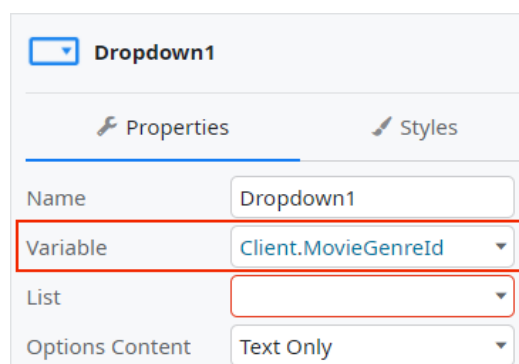


Don't forget that these visual cues are very important to the end-user of the app! Remember to always put yourself in the user's shoes.

- b. Drag a **Dropdown** widget and drop it below the Input field defined in the previous steps. Define its **width** to 9 col and **align** it to the *right*.



- c. Add a **top margin** of 20 px, under the **Styles** tab, to separate the dropdown from the input above it.
- d. Back to the **Properties** tab, set the **Variable** property of the Dropdown to *Client.MovieGenreId*.



As you see, Client Variables can be used on widgets just like a Screen's Local Variable.

- e. Notice that there are still several errors we need to solve. The first one is to set what would be the List of values that the dropdown will display.

The screenshot shows the 'Properties' tab for a widget named 'Dropdown1'. The 'Variable' is set to 'Client.MovieGenreId'. The 'List' property is highlighted with a red box, indicating it is the current focus for configuration. Other properties include 'Options Cont...' (Text Only), 'Options Text', 'Options Value', 'Mandatory' (False), 'Enabled' (True), 'Empty Text', and 'Style Classes' ('dropdown').

- f. Add an **Aggregate** to the Screen and select the **MovieGenre** Entity as the single Source.

The screenshot shows the 'GetMovieGenres' Aggregate configuration screen. The 'What data do you want to get?' section is visible, with an example text 'Example: How many orders shipped to London'. Below this, the 'Sources' section shows 'MovieGenre' as the selected source. The 'Add source' button is visible at the bottom.

- g. Back on the Screen preview, set the **List** property of the Dropdown widget to the output of the Aggregate: *GetMovieGenres.List*. All the errors will be automatically fixed.

Dropdown1

Properties **Styles**

Name	Dropdown1
Variable	Client.MovieGenreId
List	
Options Cont...	x.y Expression Editor...
Options Text	Suggestions
Options Value	GetMovieGenres.List
Mandatory	GetMovies.List

So, by recognizing the Source of the Aggregate, OutSystems infer that we want to save the Identifier of the movie genre selected and we want to display the genre's label to identify the movie genre. Which is exactly what we want!

- h. All the errors are now fixed automatically. To finish, set the **Empty Text** to *"Select a Movie Genre"*, to help the user understand what are the options within the Dropdown.

Dropdown1

Properties **Styles**

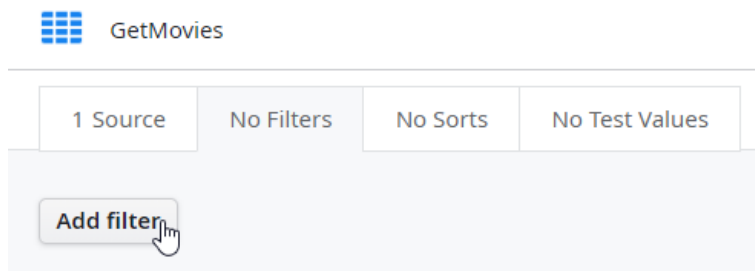
Name	Dropdown1
Variable	Client.MovieGenreId
List	GetMovieGenres.List
Options Cont...	Text Only
Options Text	MovieGenre.Label
Options Value	MovieGenre.Id
Mandatory	False
Enabled	True
Empty Text	"Select a Movie Genre"
Style Classes	"dropdown"

The Empty Text property defines what the user sees before selecting a genre.

Applying the Search

At this point, the UI is ready to apply the search filters. Now, we need to define the logic to actually apply the search on the Movies List. This has two steps: filter the **GetMovies** Aggregate results according to the search filters and trigger the search whenever the user selects the search filter.

1. We need to define two additional filters in the GetMovies Aggregate so that movies can be filtered by the search criteria chosen by the end-users (title or plot summary)..
 - a. Open the **GetMovies** Aggregate, select the **Filters** tab and click on **Add Filter**.



- b. Add the following filter condition for the title and plot summary:

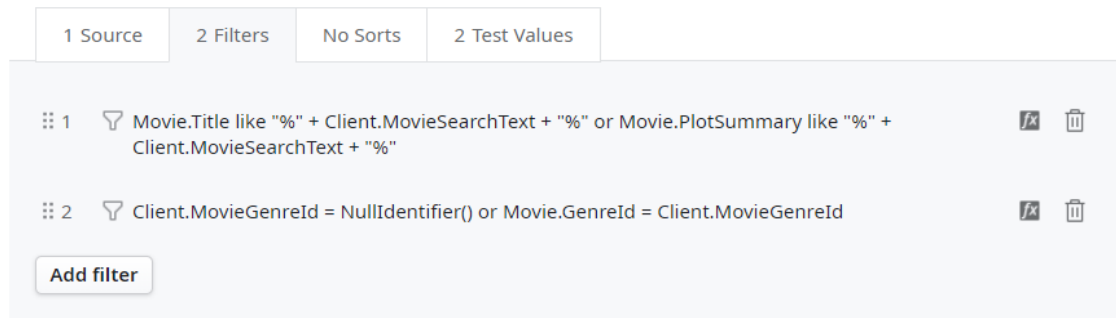
Movie.Title like "%" + Client.MovieSearchText + "%" or

Movie.PlotSummary like "%" + Client.MovieSearchText + "%"

The % are used as wildcards, so this expression can be read as: we want a movie title with some text (possibly empty), followed by the search keyword, followed by more text (also possibly empty). So, if we search for "Force", the movie Star Wars: The Force Awakens would appear in the Table since it has the keyword Force, with text before and after. The same logic is valid for the plot summary. If we didn't add the %, then the user would have to type exactly the Title or Plot Summary of the movie to find it.

- c. Add a second filter for the **Movie Genre**:

*Client.MovieGenreId = NullIdentifier() or Movie.GenreId =
Client.MovieGenreId*

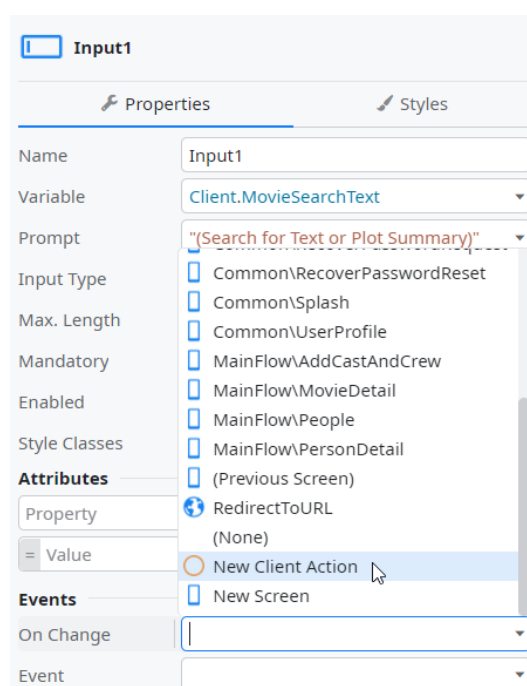


This condition is kind of special! So let's recap the desired behavior:

- If the user selects a genre, only movies of that genre should appear.
- If the user doesn't select any genre, then all movies should appear.

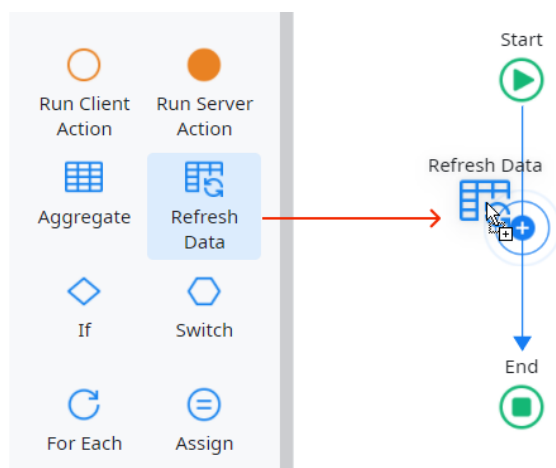
Now, consider all movies have a genre. If the condition was just *Movie.GenreId = Client.MovieGenreId*, whenever the user did not select a genre to filter the movies from, no results would be returned. Why? Because there would be no movies in the database with *GenreId = NullIdentifier()*. So, we need to consider that scenario on the Aggregate filter as well.

- Next, we need to trigger the search whenever the user types a title or plot summary or selects a movie genre in the dropdown. For that, we need to use the **OnChange** Event.
 - Select the **Input** field on the Movies Screen, expand the **OnChange** property under **Events**, and click on *New Client Action*.

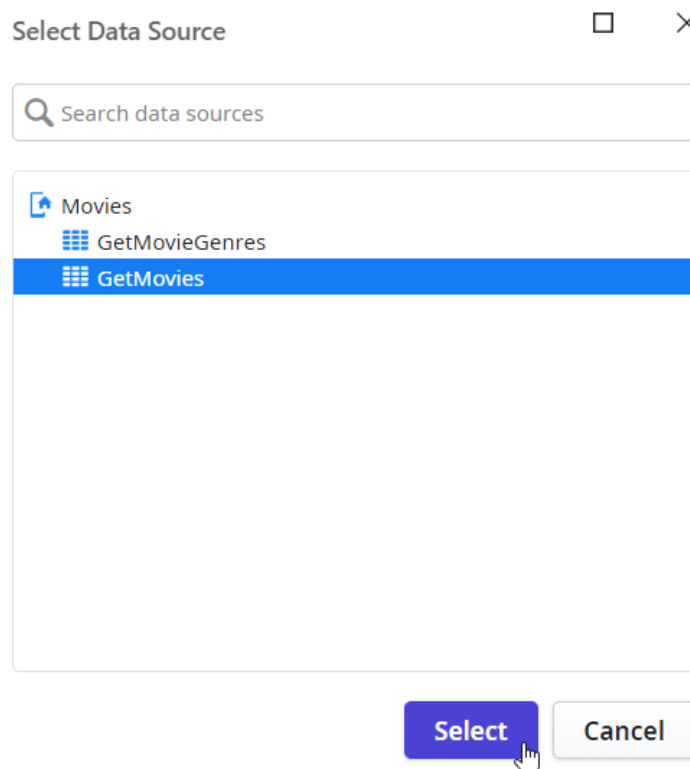


This OnChange Event is triggered automatically when the user types in an input field or selects an option from a selection-type input field, such as a dropdown for instance. This Event is really powerful since it removes the need to have a Search button. The user can just type and the logic in the Action is automatically triggered.

- b. Name the new Client Action *InputOnChange*.
- c. Drag a **RefreshData** and drop it in the Action Flow.

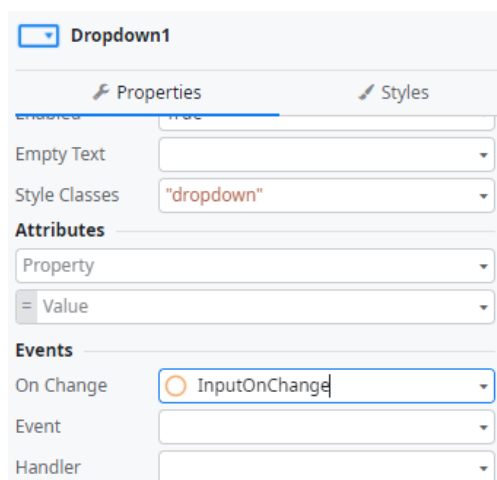


- d. Select the **GetMovies** Aggregate in the new dialog.



What we're doing here is explicitly running the GetMovies Aggregate. Why? Because the filters are already defined and the Client Variable is already filled with the user's keyword. So just by running the Aggregate again, the filter will consider that keyword and filter the results. But, why is the Client Variable already filled with a keyword? Because we're on the OnChange Event. This Action is only triggered whenever the user types or selects a keyword. Now we just need to do the same for the Dropdown.

- e. Go back to the Movies Screen and set the **OnChange** Event of the Dropdown to the same Action.



3. Publish the app and test it in the browser.



4. To properly test the behavior, we recommend these three steps:
 - a. Type a search keyword and wait for the results to appear. Do they make sense? Awesome!
 - b. Select a movie genre and wait for the results to appear. Don't forget that if you have a keyword defined in the other input field, BOTH will be applied.
 - c. Change to another Screen, navigate at will in your app and then come back to the Movies. Is the search still being applied? Nice! That's what the Client Variables do for you (well... actually one of the things it does for you!).