

# Pagination and Sorting

## Exercise - How to

|                                  |          |
|----------------------------------|----------|
| <b>Outline</b>                   | <b>1</b> |
| <b>How to</b>                    | <b>1</b> |
| Pagination                       | 1        |
| Sorting                          | 6        |
| Extra Challenges                 | 12       |
| People Screen                    | 12       |
| Sorting: Ascending vs Descending | 12       |

# Outline

In this exercise, we will add pagination and sorting functionality to the Movies Screen of the OSMDb app.

Our app can have multiple movies, which can lead to the Table in the Movies Screen having a lot of records to display. To have a better user experience, we want to add the pagination functionality so that each page only shows five movies at a time.

Then, we want to enable the users to dynamically sort the movies by clicking on any column header in the Table. This means that if the user clicks on the **Title** header, the movies should appear sorted by movie title.

In the end, we have some extra challenges where you'll implement two scenarios:

- When the user clicks on a column header twice in a row, the order of the sorting should change from ascending to descending.
- When the sorting by a column is done, the pagination should reset and go back to the first page.

Let's do it!

## How to

In this section, we'll describe exercise *11 - Pagination and Sorting*, step by step.

### Pagination

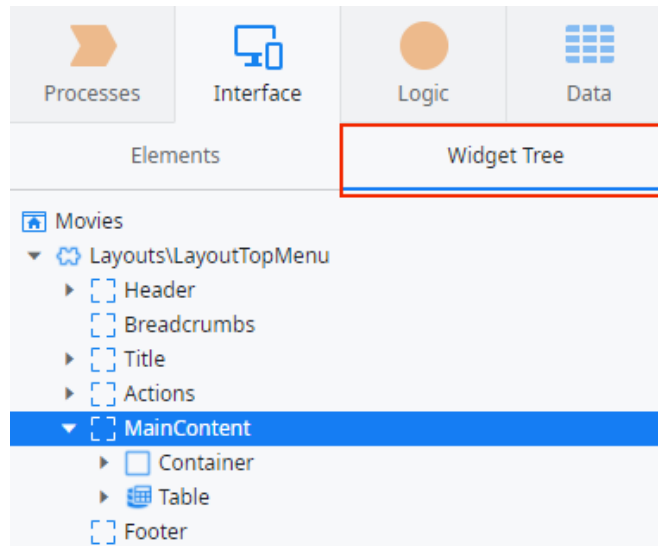
First, we will implement the pagination functionality on the Movies Table, using a **Pagination** widget. With that, we want to display just five movies per page. We'll do this in 3 steps:

- Add the Pagination widget to the Movies Screen.
- Create the logic to fetch the next records to be displayed.
- Adjust the Aggregate to the Pagination functionality, to avoid returning all records at once.

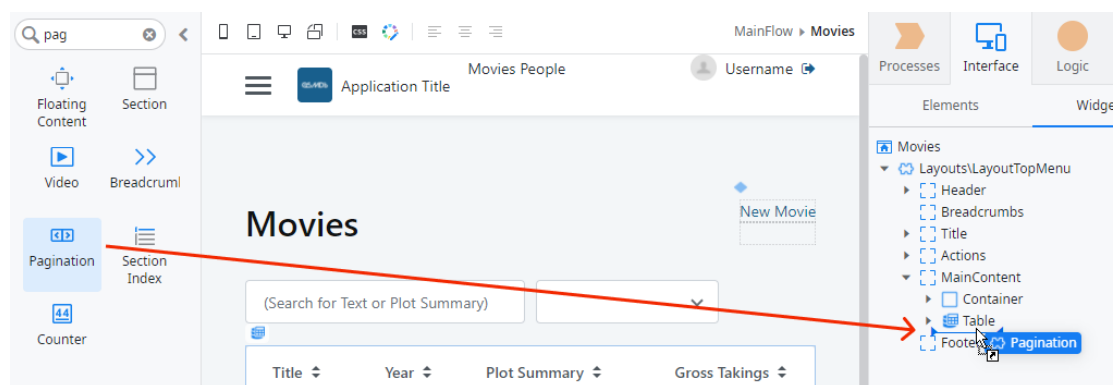
Let's start with the first step then!

1. Add the **Pagination** widget to the Movies Screen **right after** the Movies table. Create two Local Variables to represent the Start Index of each page and the maximum number of movies we want to display per page (MaxRecords).

- a. On the Movies Screen, open the **Widget Tree** and expand the MainContent area.



- b. In the widget toolbar on the left, find the **Pagination** widget, drag it and drop it after the Table.



The Pagination has four properties that have an error at this point: **StartIndex**, which represents the first record on each page; **MaxRecords**, which represents the number of records on each page; **TotalCount**, which represents the total number of records displayed on the Table; Event **Handler**, which expects an Action that will be triggered whenever the user changes page.

- c. Create a Local Variable named *MaxRecords*. Its **Data Type** should be set to *Integer* and the **Default Value** 5.

The screenshot shows the configuration panel for a local variable named **MaxRecords**. It has a yellow square icon and is labeled "Local Variable". The fields are: Name (MaxRecords), Description (empty), Data Type (Integer), and Default Value (5).

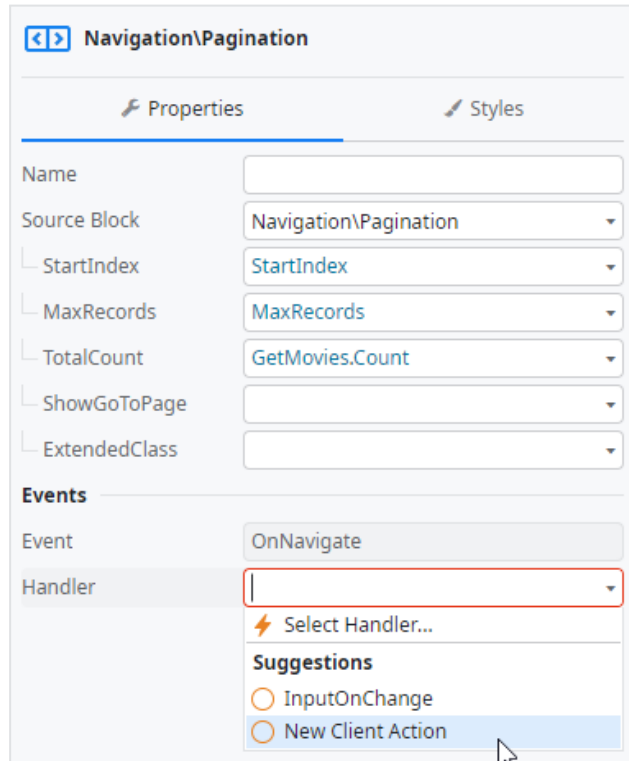
- d. Create another Local Variable called *StartIndex*. Its **Data Type** should be set to *Integer* and the **Default Value** 0.

The screenshot shows the configuration panel for a local variable named **StartIndex**. It has a yellow square icon and is labeled "Local Variable". The fields are: Name (StartIndex), Description (empty), Data Type (Integer), and Default Value (0).

- e. Select the Pagination widget by clicking on it, and set **StartIndex** and **MaxRecords** properties to the respective Local Variables. Also, set the **TotalCount** to *GetMovies.Count*, which gives us the number of records returned by the GetMovies Aggregate.

The screenshot shows the configuration panel for the **Navigation\Pagination** widget. It has a blue icon with left and right arrows. The panel has two tabs: **Properties** and **Styles**. Under the **Properties** tab, the fields are: Name (empty), Source Block (Navigation\Pagination), StartIndex (StartIndex), MaxRecords (MaxRecords), TotalCount (GetMovies.Count), ShowGoToPage (empty), and ExtendedClass (empty). Under the **Events** tab, the fields are: Event (OnNavigate) and Handler (empty).

2. Create the logic to support the behavior when a user changes the page. The Pagination has a Handler that should trigger a Client Action. Within that Action, we should implement the logic to change the Start Index and fetch the records for that page.
  - a. In the same property dialog, expand the **Handler** property and click on **New Client Action**. This Action will have the logic that supports the change of page.



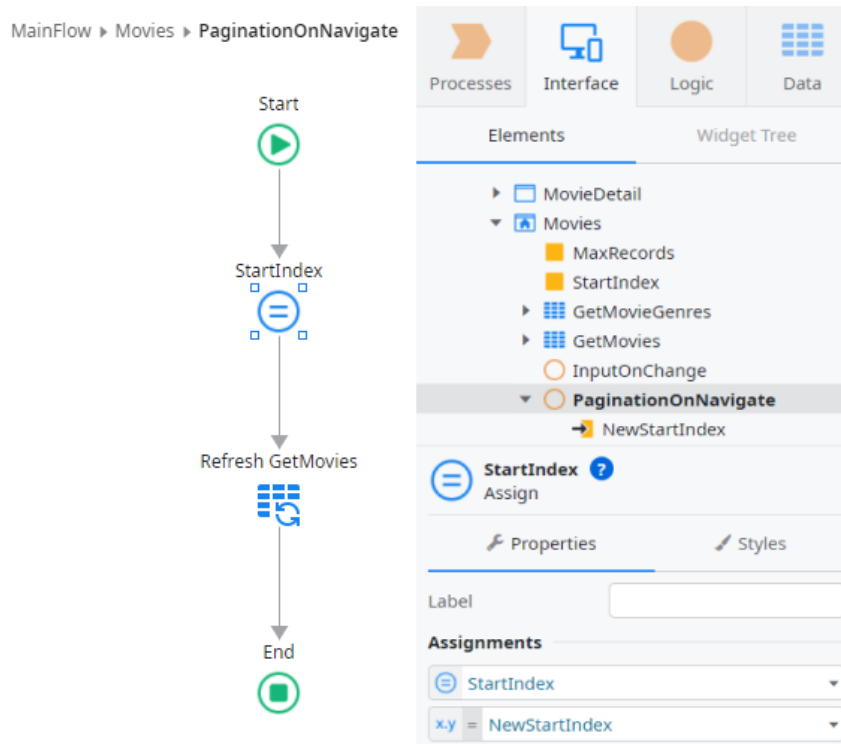
A new Action named **PaginationOnNavigate** is created with an Input Parameter: **NewStartIndex**. When the user clicks on a new page to see other movie records in the Table, this Input will pass to the Action the information about the Start Index of the page clicked.

- b. Drag an Assign and drop it on the Action flow. Set the assignment to be:

*StartIndex = NewStartIndex*

With this assignment we're updating the StartIndex Local Variable, which is being used by the Pagination widget. This will help the widget act accordingly.

- c. Drag a **Refresh Data** node and drop it on the Action flow. In the new dialog, select the **GetMovies** Aggregate. The Action flow should look like this:



The Refresh Data node will trigger the execution of the GetMovies Aggregate again. The idea is to get the records that correspond to the page that was selected. However, refreshing the Aggregate is not enough. To make this work, we need to modify the Aggregate to only return the number of records per page (MaxRecords), starting from a particular record instead of always from the beginning (StartIndex).

3. Let's then change the **GetMovies** Aggregate to consider the StartIndex and the MaxRecords values. Just click on the Aggregate and set the properties to the respective Local Variables.

The screenshot shows the 'GetMovies Aggregate' properties dialog. It has the following fields and values:

| Property               | Value                    |
|------------------------|--------------------------|
| Name                   | GetMovies                |
| Description            |                          |
| Server Request Time... | (Module Default Timeout) |
| Start Index            | StartIndex               |
| Max. Records           | MaxRecords               |
| Fetch                  | At start                 |

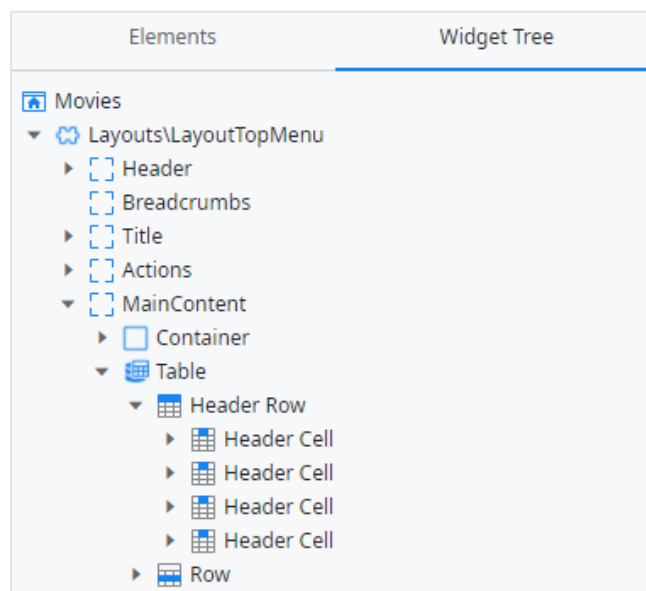
4. Publish the app and test the pagination in the Movies Screen. Make sure that the records change accordingly.



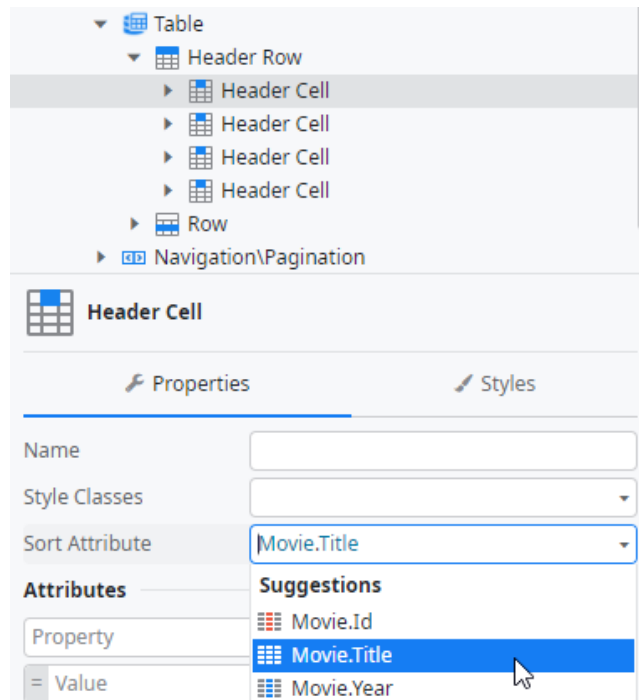
## Sorting

In the second part of this exercise, we will implement dynamic sorting on the Movies Table. Basically, whenever a user clicks on the header of a column of the Table, the results will automatically appear sorted by the values in that column.

1. In the Movies Screen, define the Sort Attribute of each Header Cell to the respective Entity attribute.
  - a. Open the Widget Tree on the Movies Screen. Expand the MainContent, then the Table, and finally the **Header Row** until you see the **Header Cells**.



- b. For each Cell, the **Sort Attribute** property should be set as the corresponding Entity attribute. For instance, the first Header Cell corresponds to the Title, so the **Sort Attribute** should be set to *Movie.Title*. If this is not the case, set it manually.

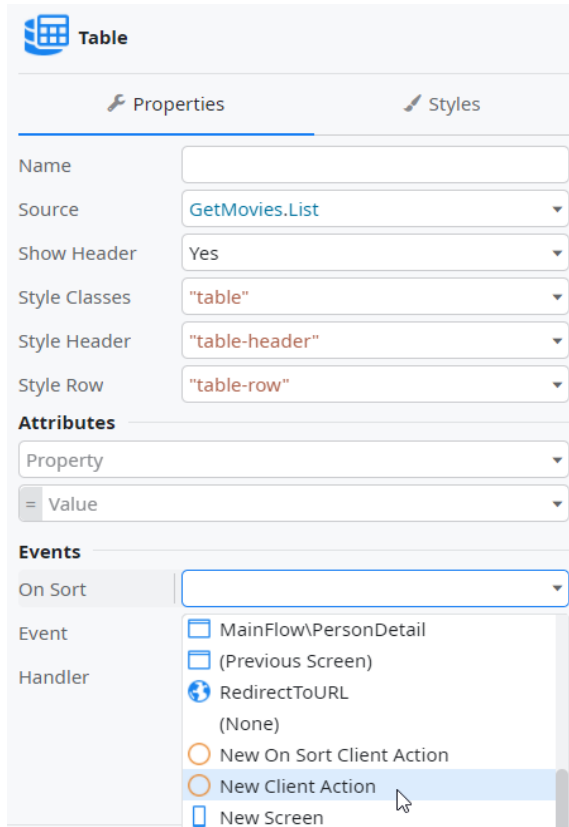


Now that the header cells have the attributes assigned to them, we need to work on the logic to trigger the sorting.

2. Set the **On Sort** Event of the Table to a new Client Action, where the logic for the sort will be implemented. This logic will require the sorting criteria to be stored in a local variable. Then, the start index should be reset and the Aggregate refreshed.



- a. Select the Table and in the **On Sort** Event choose **New Client Action**. This will create an OnSort Action.



The screenshot shows the configuration panel for a Table widget. The 'Events' tab is active. Under the 'On Sort' event, a dropdown menu is open, displaying a list of possible handlers. The 'New Client Action' option is highlighted by the mouse cursor.

| Property      | Value          |
|---------------|----------------|
| Name          |                |
| Source        | GetMovies.List |
| Show Header   | Yes            |
| Style Classes | "table"        |
| Style Header  | "table-header" |
| Style Row     | "table-row"    |

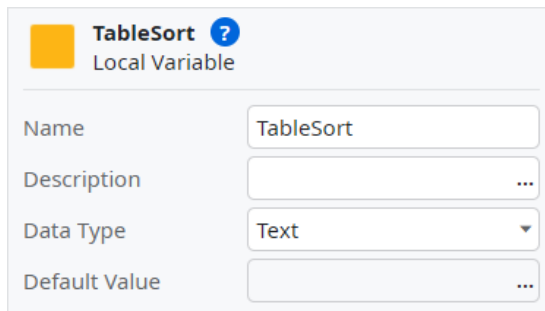
**Attributes**

Property:   
= Value:

**Events**

On Sort:   
Event:   
Handler:   
MainFlow\PersonDetail   
(Previous Screen)   
RedirectToURL   
(None)   
New On Sort Client Action   
New Client Action   
New Screen

- b. Create a **Local Variable** on the Movies Screen called *TableSort*, with **Data Type** set to *Text*.

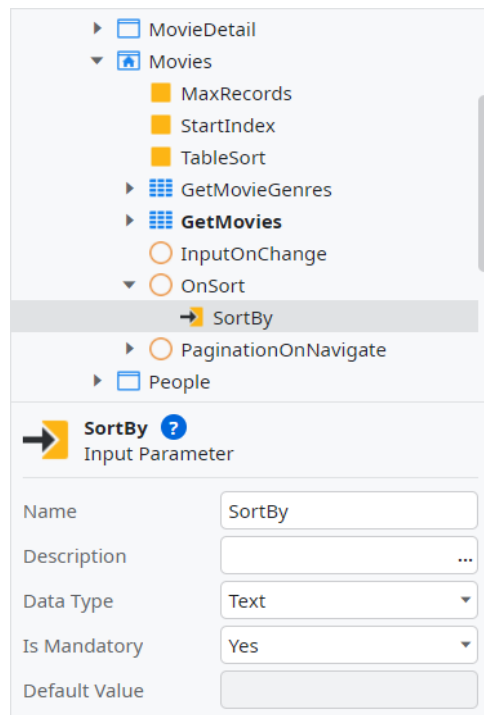


The screenshot shows the configuration panel for a Local Variable named 'TableSort'. The 'Data Type' is set to 'Text'.

| Property      | Value     |
|---------------|-----------|
| Name          | TableSort |
| Description   |           |
| Data Type     | Text      |
| Default Value |           |

This Local Variable will hold the choice made by the user for the sorting criterion.

- c. Add a new **Input Parameter** to the *OnSort* Action called **SortBy** with **Data Type** set to **Text**.



Just like it happened with the pagination, this Action also includes an Input Parameter that holds the sort by attribute, meaning the column that the user clicked on.

- d. Drag an **Assign** and drop it on the *OnSort* Action flow. Define the following assignments:

*TableSort* = *SortBy*

*StartIndex* = 0

The 'Assign' dialog is shown with the following configuration:

- Label:** (Empty text field)
- Assignments:**
  - Row 1: Variable `TableSort` assigned to `SortBy`.
  - Row 2: Variable `StartIndex` assigned to `0`.

We are setting the `TableSort` variable to the value passed as the input parameter. We will come back to this later. The `StartIndex` is set to 0 so that after sorting, we come back to the first page of the Movies Table pagination.

- e. Drag a **Refresh Data** node and drop it after the Assign. Select the **GetMovies** Aggregate in the new dialog.
- f. To finish the logic, go back to the Movies Screen, select the Table and notice there is an error on the input parameter of the OnSort Action.

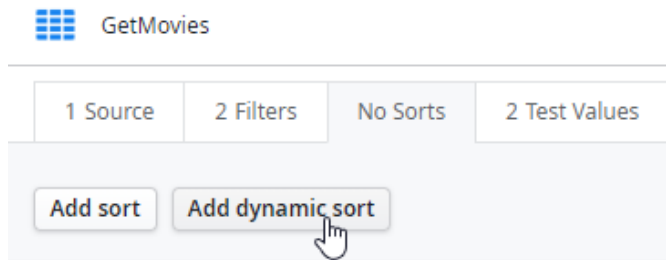
The error bar displays the following message:

1 Error | Debugger | 1-Click Publish

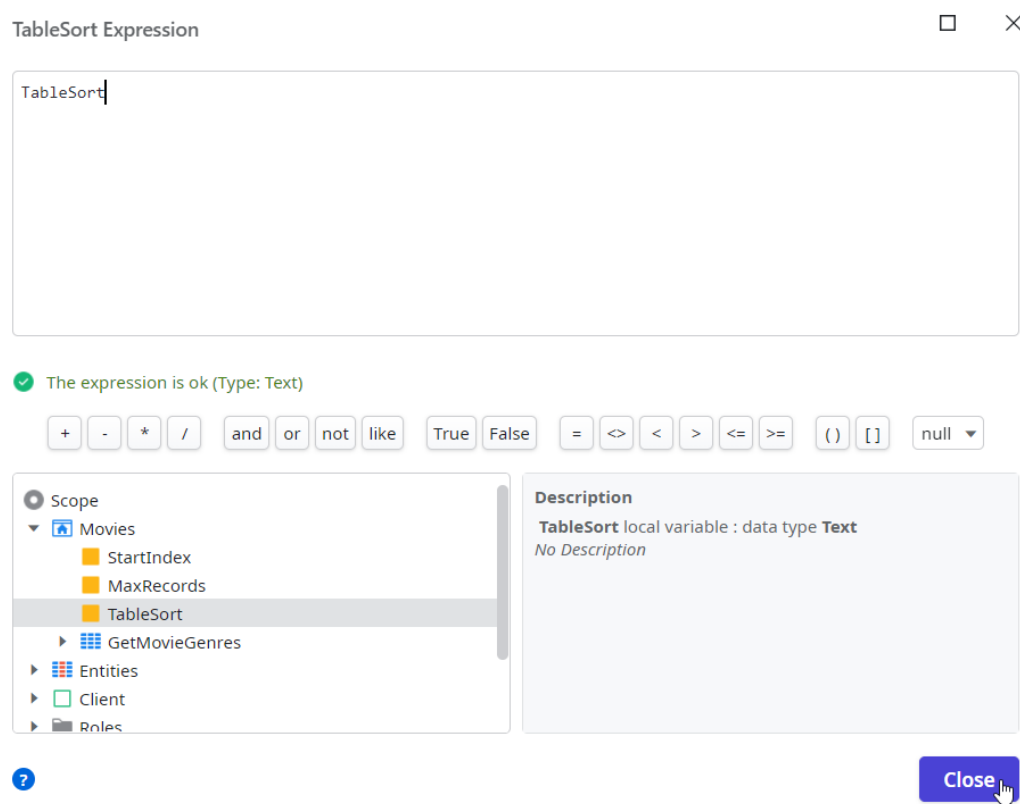
Required Property Value | A valid expression must be set for parameter 'SortBy'.

We're missing the value for the `SortBy` attribute. Set the value of the input parameter to `ClickedColumn`. The `ClickedColumn` input appears automatically when we define the On Sort Event Action handler. This `ClickedColumn` value will automatically hold the selection that the end-user makes during runtime, meaning the Sort Attribute of the column selected. Then, in the Action, we save this information in the `TableSort` Local Variable.

3. At this point, all the logic is set, but there is an important step missing. The Aggregate needs to be adjusted so that it fetches the data according to the sorting criteria selected by the user. To support that, we need to add a dynamic sorting to the Aggregate.
  - a. Open the **GetMovies** Aggregate and select the **Sorting** tab.
  - b. Click on the option **Add Dynamic Sort**



- c. Set the Expression to *TableSort*. Click **Close** to confirm.



The **TableSort** Local Variable has the sorting criteria selected by the user, so by setting the dynamic sort to the value of the local variable, the Aggregate will be sorted by the selected criteria. The **Add Sort** option could not be chosen in this case since it just implements static sorting, meaning you need to choose an attribute and the sorting is just performed for that attribute. The **Add Dynamic Sort** is the option to go when implementing dynamic sorting on a Table / List

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



And that's it! You have your very first app finished. If you feel like it, check out the extra challenges to expand the pagination and sorting functionality.

## Extra Challenges

If you have some time, there are some extra challenges we can do.

### People Screen

Apply the same logic sorting logic to the People Screen but this time use the OutSystems accelerator, selecting the **New OnSort Client Action** option in the Table.

Also, you can follow the same strategy used for the movies and create the pagination for the people records.

### Sorting: Ascending vs Descending

At this point, the sorting logic is prepared to sort the records following an ascending order. If you performed the previous challenge, you can see that the logic is a bit more complex, to support the descending order.

If not, the objective is this: when the user clicks on a column header for the first time, the records should be sorted in ascending order by the attribute that corresponds to that column. However, when the user clicks on the same header column again, the sorting order should change to descending. To do so, you should use the TableSort variable and the **DESC** keyword to add to the sorting criterion.