

Blocks and Events

Exercise - How to

Outline	1
How to	1
Create Thumbs Block	1
Display the Block in the MovieDetail Screen	5
Rating a Movie	10
Permissions to Rate a Movie	17
Final Touches	19

Outline

In this exercise, we will introduce Blocks in our app to allow users to like or dislike a movie. The rating will be based on choosing thumbs up for “like” and thumbs down for “dislike”. This functionality should be implemented in a Block and then used in the MovieDetail Screen.

To accomplish this objective, we will:

1. Create a new Block.
2. Use the thumbs up and thumbs down icons (filled and hollowed) to display the several options the user has to choose from.
3. Display the thumbs up/down information in the MovieDetail Screen.
4. Make the icons “clickable” and trigger the logic to create/update the rating for that particular movie/person.
5. Make sure the rating chosen is recorded in the database.
6. Make sure the thumbs are only clickable by users with the OSMDbUser Role.

A user can rate the same movie more than once. In those cases, the application should consider this as an update of the previous rating, and not as a new rating from that user.

How to

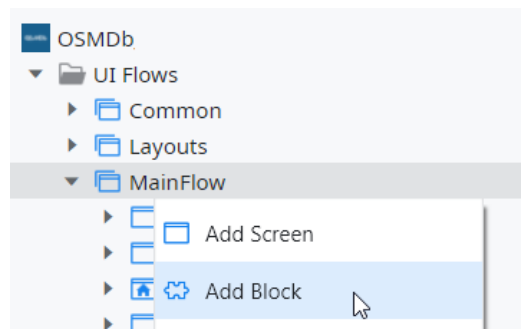
In this section, we'll describe exercise 8 - *Blocks and Events*, step by step.

Create Thumbs Block

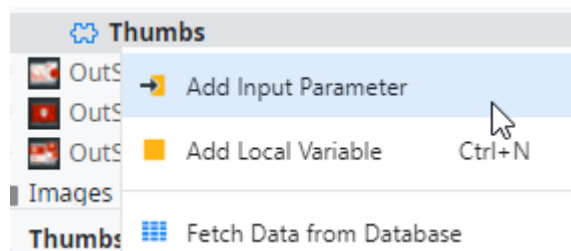
Let's start by creating the Thumbs Block. This Block will be created in the MainFlow, next to the other Screens. The Block should have four icons: two for the thumbs up (filled and hollowed) and two for the thumbs down (filled and hollowed). Why four? Because if we click on the thumbs up icon, that one should become filled and the thumbs down hollowed. If we click on it again, then both should become hollow. And so on. To control the icons, we need two Boolean input parameters in the Block: *Rating* (True = thumbs up and False = thumbs down) and *IsRated*, to distinguish if we already rated the movie or not.

1. Create a new Block and call it *Thumbs*. Create two Input Parameters: *Rating* (Boolean) and *IsRated* (Boolean).

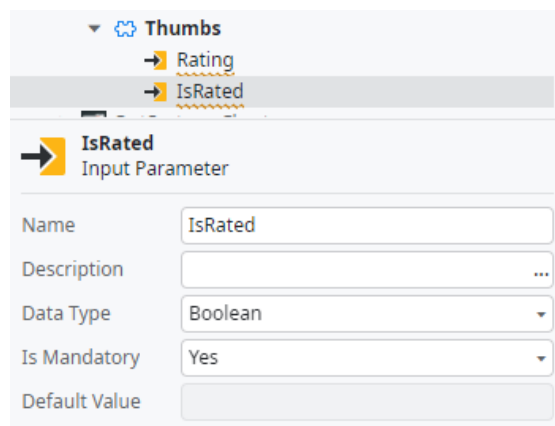
- a. Under the Interface tab, right-click on the **MainFlow** and select **Add Block**.



- b. Set the **Name** of the Block to *Thumbs*.
- c. Just like with Screens, right-click on the Block and select **Add Input Parameter** to add a new input.



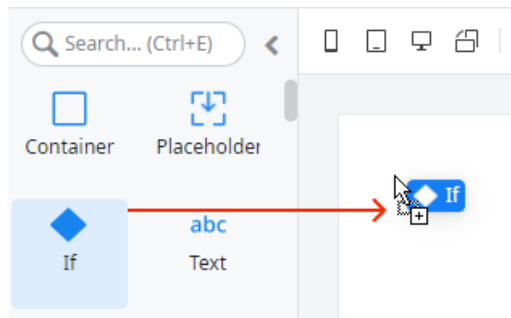
- d. Set its **Name** as *Rating* and the **Data Type** as *Boolean*.
- e. Add another **Input Parameter** and call it *IsRated*. This should also be a *Boolean*.



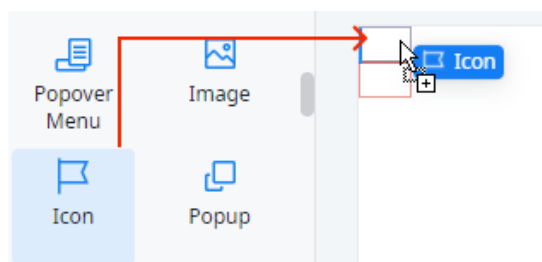
2. Now it's time to implement the UI of the icon. The Block will show thumbs up and thumbs down icons. To distinguish if they are selected or not, we will have two versions of each one: a filled and a hollowed icon. The icons that will appear to the end-user

depend if the movie is rated or not (IsRated) and the Rating given. But we'll work on the behavior later.

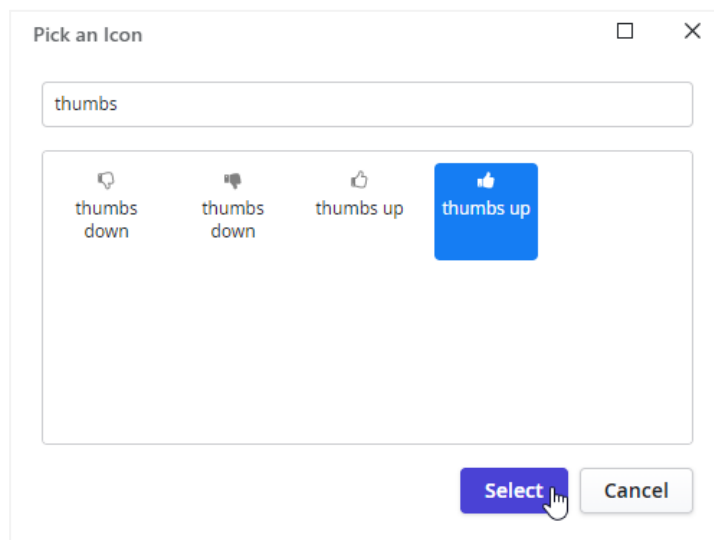
- a. Drag an **If** and drop it inside the Block.



- b. Drag an **Icon** inside the **True** branch of the If.

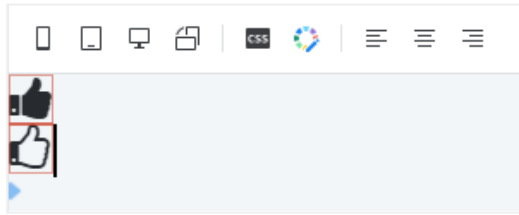


- c. In the new dialog, select the filled **thumbs up** icon.

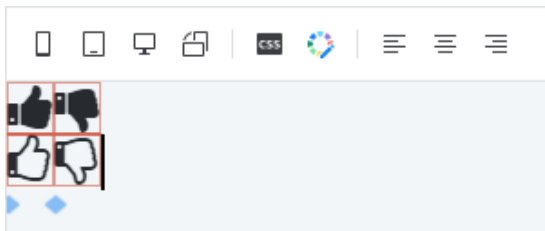


Note: There are 2 options for the thumbs up and 2 for the thumbs down icons. One will be used for the clicked thumb and one for when it is not clicked (hollow).

- d. Repeat the same process for the **False** branch, but this time choose the **hollow thumbs up** on the icons. Your result should look like this:



- e. Drag another if next to the previous one and repeat the steps with the thumbs-down icons. Your result should look like this:

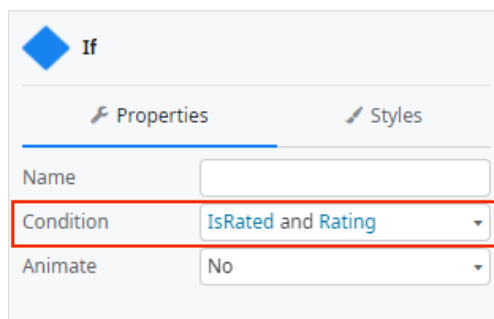


3. Now that we have the UI, we need to define the logic to make sure the right icons appear when they should. So, let's see the options we have:

- Thumbs Up (Filled): Should appear when the movie is rated ($IsRated = True$) and when the Rating is positive ($Rating = True$)
- Thumbs Up (Hollow): Should appear whenever the Condition above is not met.
- Thumbs Down (Filled): Should appear when the movie is rated ($IsRated = True$) and when the Rating is negative ($Rating = False$).
- Thumbs Down (Hollow): Should appear whenever the Condition above is not met.

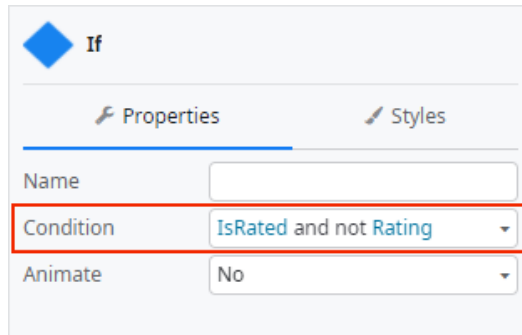
This can be done by simply defining the right conditions in both Ifs.

- a. Select the **first** If and set its **Condition** to be: *IsRated and Rating*



This means the thumbs-up icon will only appear filled if the variable *IsRated* is true AND the variable *Rating* is also true. Remember, both conditions must be met! In any other case, the hollow thumbs up icon will appear.

- b. Select the second If and set its **Condition** to be: *IsRated and not Rating*



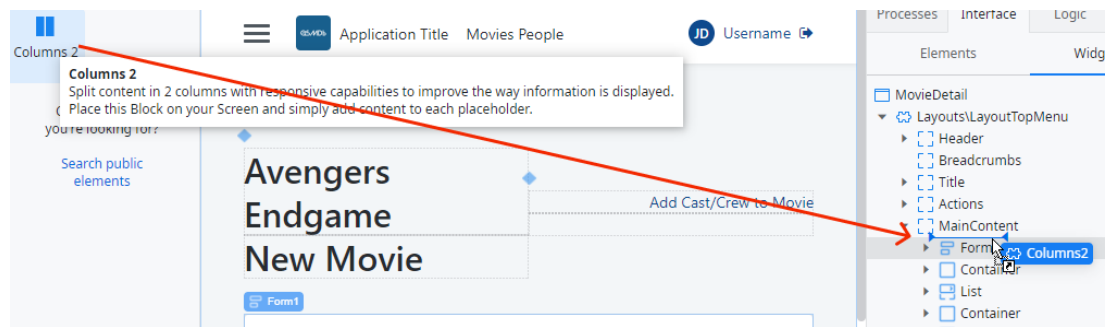
Just like in the previous If, the thumbs down icon should appear filled only if *IsRated* is true. The difference here is the *Rating* value, that in this case, should be false (since it's a thumbs down).

Display the Block in the MovieDetail Screen

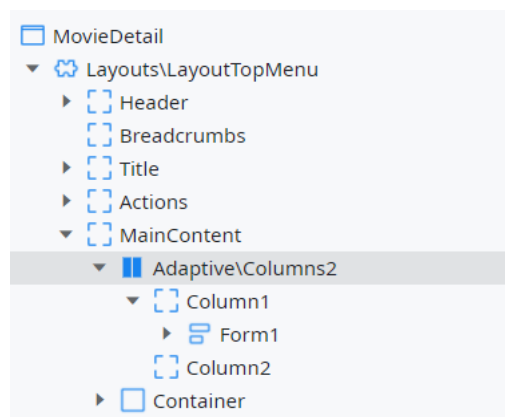
Now that we have the Block implemented, let's use it in the MovieDetail Screen. First, we need to save some space for the Block, by splitting the first part of the MovieDetail Screen into two columns: the first column for the Form and the second column for the rating.

1. Let's create some space for the Block in the MovieDetail Screen. Split the Screen into 2 columns. The first column will have the movie Form and the second column will have a subtitle with the text: *Your Rating*.
 - a. In the MovieDetail Screen, open the Widget Tree and expand the **MainContent** placeholder to see its elements.

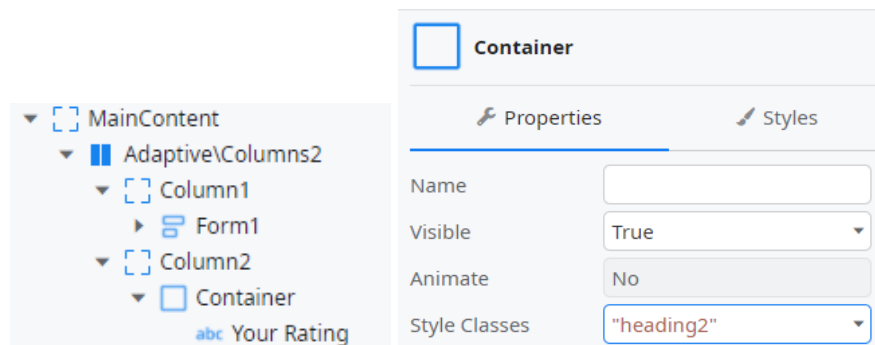
- b. Drag a **Columns2** widget and place it on the MainContent, right before the Form.



- c. Now, place the **Form** inside the **Column1** section of the recently added widget.



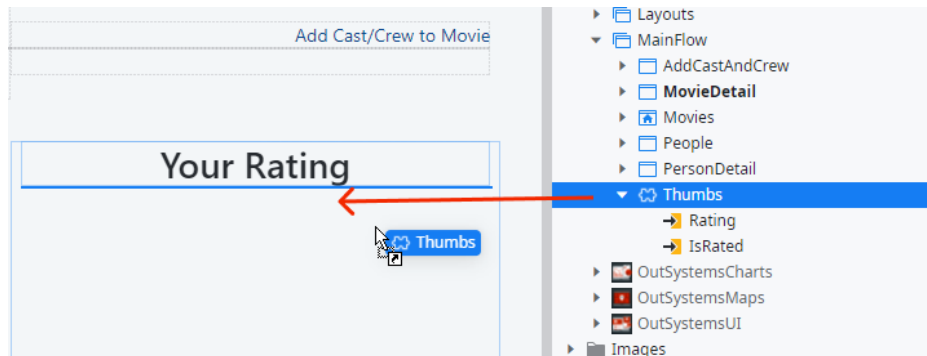
- d. Drag a **Container** and place it in **Column2** content. Type *Your Rating* and set the text **Style Classes** property to "heading2".



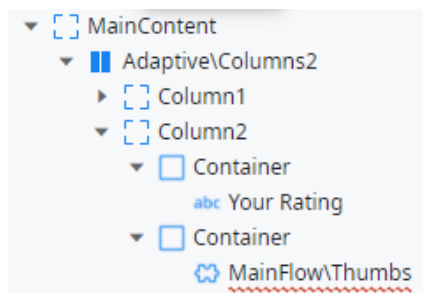
- e. Center the text inside the Container, using the options on top of ODC Studio.



2. Place the **Thumbs** Block in the second column of the MovieDetail Screen, right below the **Your Rating** text. Set the Inputs of the Block with the default values.
 - a. Drag the **Thumbs** Block and drop it right below the “Your Rating” text.

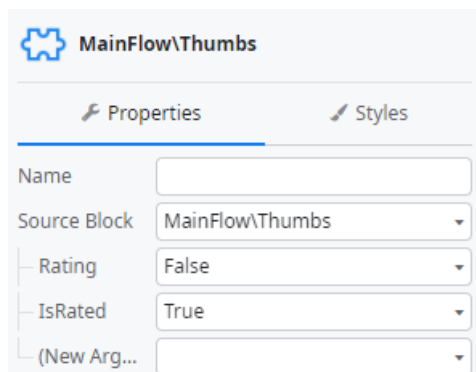


- b. Enclose the Block in a **Container** and make sure the thumbs appear centered in the column. Your Widget Tree should look like this:



At this point, the Block has an error. This error is caused by the two Input Parameters of the Block. Since the Input Parameters are mandatory they expect a value.

- c. In the MovieDetail screen, select the *Thumbs* block, go to its properties, and set the **Rating** value to *False* and the **IsRated** value to *True*.



This is just for testing purposes at this point. We'll provide different values later in the exercise.

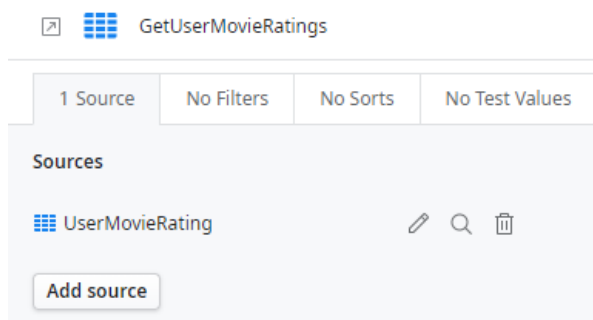
- d. Publish the app and make sure the thumbs appear properly on the browser.



You should see a hollow thumbs up and a filled thumbs down.



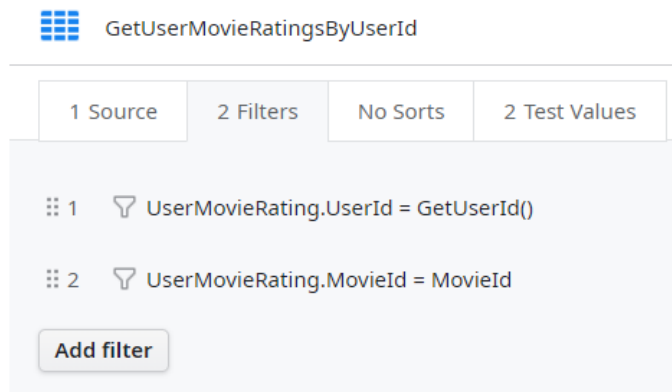
3. Like we mentioned in the previous steps, we just assigned a static value to the Block's Input Parameters for testing purposes. We don't want that in our app! So, we need to create a new Aggregate to fetch the Movie's real rating and if it is rated or not, and display the proper information on the Screen through the Thumbs Block. Where is this information saved? In the UserMovieRating Entity we created a couple of exercises ago.
 - a. Right click on the MovieDetail Screen and choose **Fetch Data from Database**. In the Aggregate, set the **UserMovieRating** as the Source.



b. Create two new filters in the Aggregate:

- *UserMovieRating.UserId = GetUserId()* - to get the rating for the user logged in, using the *GetUserId()* function.
- *UserMovieRating.MovieId = MovieId* - to guarantee that we're fetching the rating for the movie being displayed in the MovieDetail Screen.

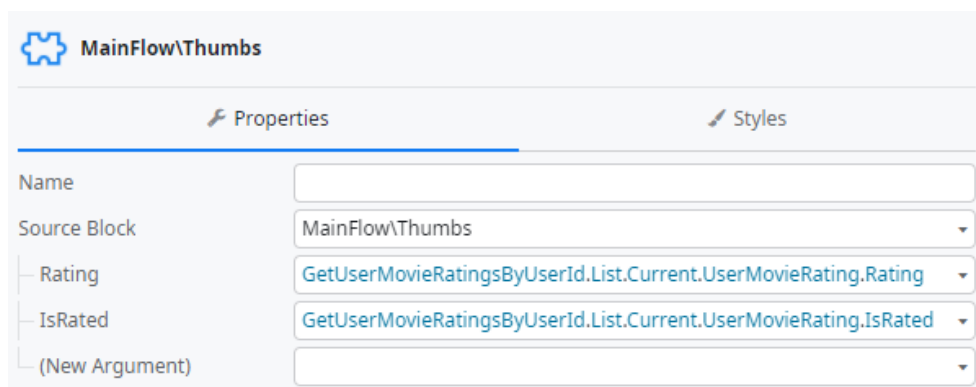
Then, update the name of the Aggregate to *GetUserMovieRatingsByUserId* if it is not automatically updated.



c. Go back to the MovieDetail screen, select the **Thumbs** Block to see its properties, then set the following values:

Rating -> *GetUserMovieRatingsByUserId.List.Current.UserMovieRating.Rating*

IsRated -> *GetUserMovieRatingsByUserId.List.Current.UserMovieRating.IsRated*



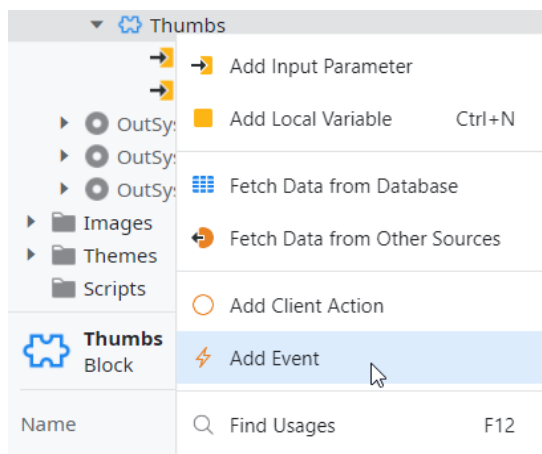
The Expressions are quite long, but they are very similar to what we've been doing so far. The Aggregate will fetch the Rating and IsRated information for the logged-in user for this specific movie from the UserMovieRating Entity. And that's precisely the values that the Block is expecting!

Rating a Movie

Now, let's implement the "clickable" scenario. Each icon will have a Link that should trigger an Event, *ThumbClicked*. The Event will be used to send information to the parent, namely the selected rating and if the movie is now rated, or if we just removed the previous rating given to the movie.

When the Event is triggered, the parent will then handle it by automatically executing a Client Action. This Action will then have the necessary logic to save the rating information in the *UserMovieRating* Entity. Finally, the rating should only be saved in the database if the user has the *OSMDBUser* Role.

1. In the **Thumbs** Block, enclose every icon in Links. All the Links should trigger an Event, *ThumbClicked*, that will send the Rating and IsRated values to the parent of the Block, which in this case is the *MovieDetail* Screen.
 - a. Right-click on the **Thumbs** Block and select **Add Event**.

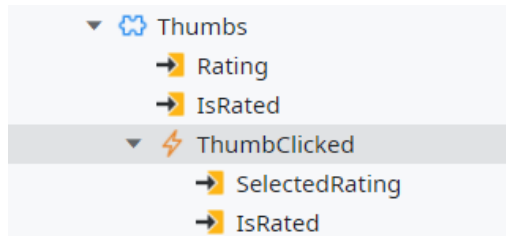


- b. Set the **Name** of the Event to *ThumbClicked* and make it non-mandatory.

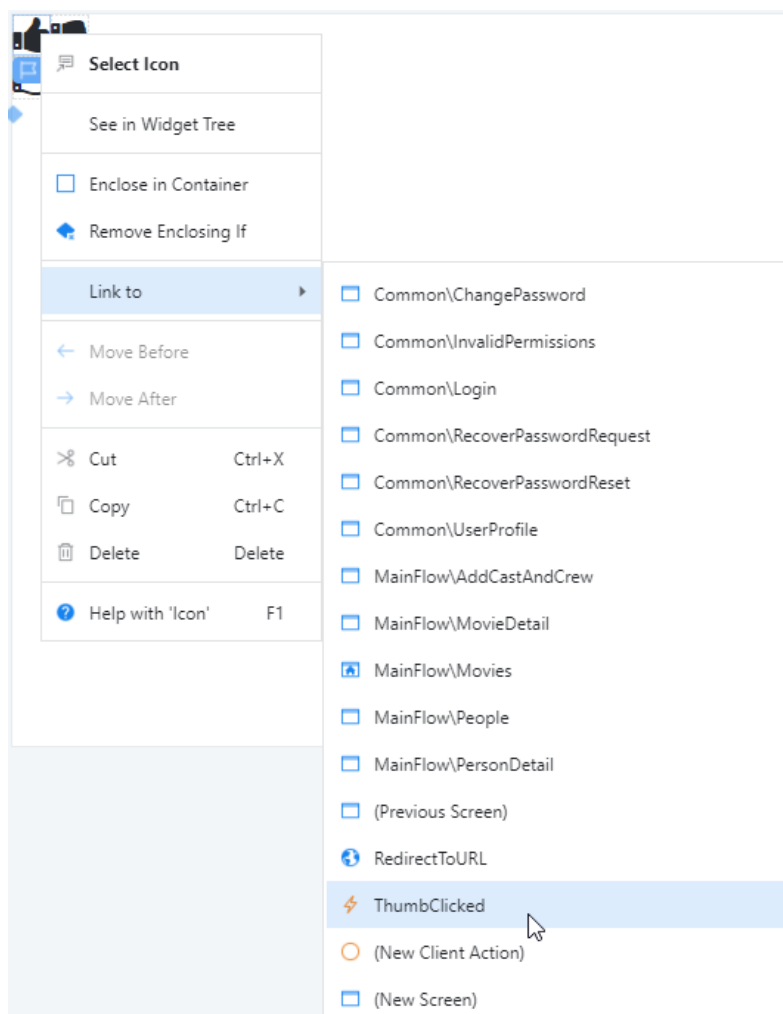
A screenshot of the 'ThumbClicked' Event configuration form. The form has a title 'ThumbClicked Event' with a lightning bolt icon. It contains three fields: 'Name' with the value 'ThumbClicked', 'Description' which is empty, and 'Is Mandatory' with a dropdown menu set to 'No'.

The Is Mandatory property of an Event is important. If the Event is mandatory, the parent is forced to handle the notification it receives when the Event is triggered. When the Event is not mandatory, then it's an optional scenario.

- c. Add an Input Parameter to the Event and name it *SelectedRating*, with the **Data Type** set to *Boolean*. Add another Input Parameter and name it *IsRated*, with the **Data Type** also set to *Boolean*.



- d. Double-click the **Thumbs** Block to open it in the preview area. Right-click the first thumbs up icon and elect **Link to -> ThumbClicked**



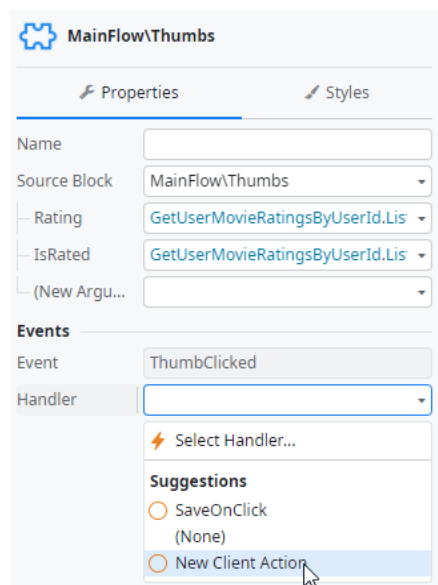
Note: We can define a Link to trigger an Event. In that case, the Event will be triggered when the end-user clicks on the Link.

- e. Repeat the same step for all the icons. Every single one of them will trigger the same Event.
- f. Now, all four Links have errors. That's because the Event has input parameters, and they are expecting a value for each one of them. This value depends on the icon that is clicked, so the Links will have as values what follows:
 - Filled Thumbs Up Icon: *SelectedRating* = True and *IsRated* = False
 - Hollow Thumbs Up Icon: *SelectedRating* = True and *IsRated* = True
 - Filled Thumbs Down Icon: *SelectedRating* = False and *IsRated* = False
 - Hollow Thumbs Down Icon: *SelectedRating* = False and *IsRated* = True

Don't forget that these values are the ones that will become effective when the user clicks on an icon. For instance, if the thumbs up icon is filled, it means we gave a positive rating and the movie is rated. If we click on it again, it will become hollow, turning the Rating and the IsRated to False.

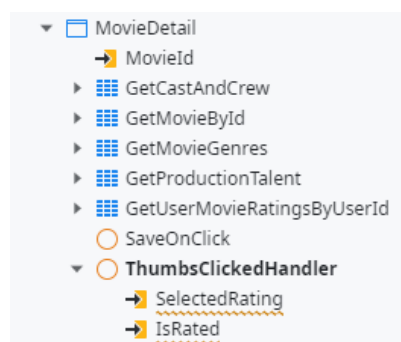
2. Since we have the Block triggering a non-mandatory event, now the parent may or may not respond to that event. However, in this case, we want the Screen to respond to the Event. Why? Because we want to save the Rating and IsRated information in the database. So, let's start by setting up the Event Handler.
 - a. Open the **MovieDetail** Screen and select the **Thumbs** Block to see its properties.

- b. In the **Events** section of the properties, there's a **Handler** property. Set this property to a **New Client Action**.



So, what's happening here? We're saying what will happen on the Screen when the Event is triggered. So, when a user clicks on an icon, the Block triggers the event and automatically this new Action will be executed by the Screen.

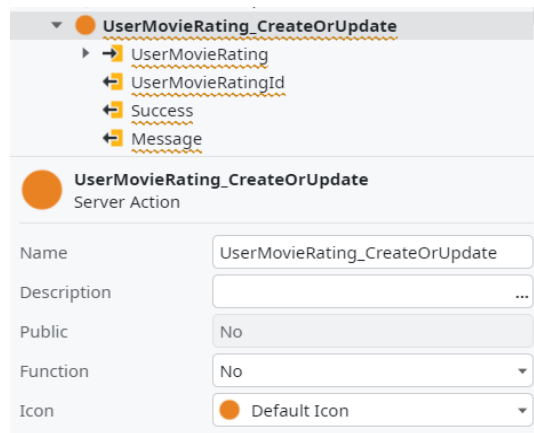
- c. Set the **Name** of the new Client Action to *ThumbsClickedHandler*. Notice that two input parameters were automatically created: *SelectedRating* and *IsRated*.



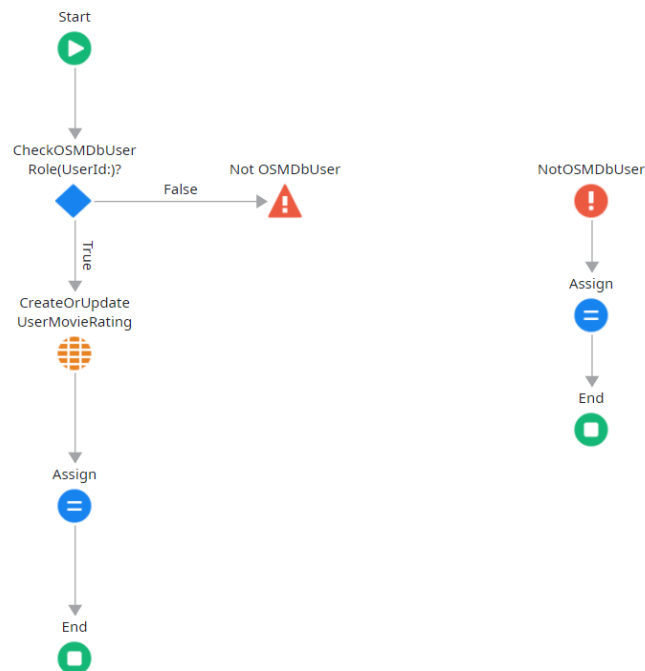
The Input Parameters were automatically created because this Action was created under the scope of an Event handler.

3. Now, we need to save the movie ratings in the database. Just like we did for movies and people, we will have server-side and client-side logic. Let's start with the Server Action, which will follow a very similar approach to the other Server Actions created in previous exercises.

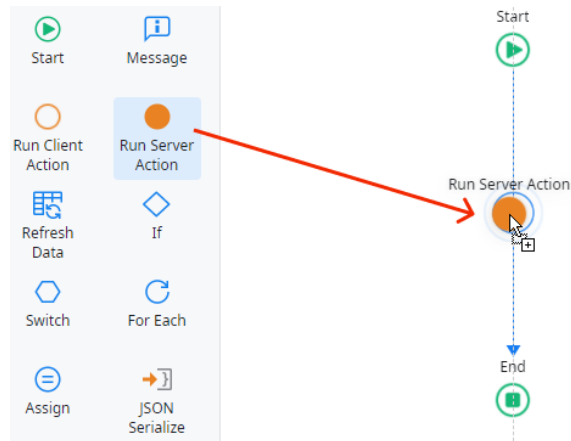
- a. Under the Logic tab, create a new **Server Action** called *UserMovieRating_CreateOrUpdate*. Add one Input Parameter called *UserMovieRating* and three Output Parameters, *UserMovieRatingId* (UserMovieRating Identifier), *Success* (Boolean), and *Message* (Text).



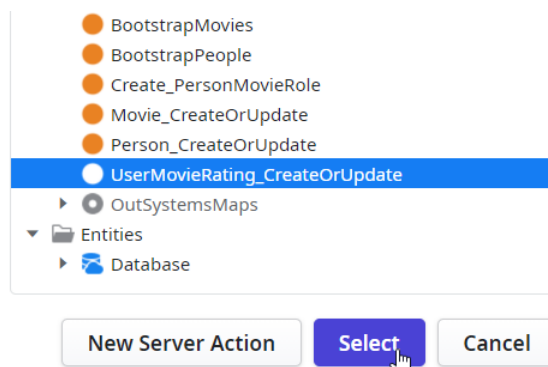
- b. The logic should be similar to the other Server Actions created for movies, people, and cast/crew. The main difference is that this operation will be available to users with the **OSMDBUser** Role. If the check for the role is successful, the info is saved in the database, the Success parameter must be set to True and the UserMovieRatingId parameter to the identifier of the record added/changed. Otherwise, an exception is raised, the Success is set to False and an error Message should be set. The logic should look like the following screenshot:



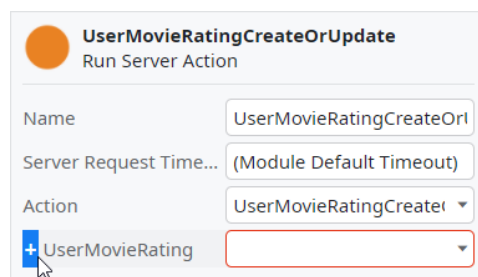
4. Now, we can implement the client-side logic. It will call this new Server Action and pass the right information to be saved in the database.
 - a. Switch back to the Interface tab and find the **ThumbsClickedHandler** Client Action under the MovieDetail Screen.
 - b. Drag a **Run Server Action** from the left sidebar and drop it on the Action flow.



- c. Select the **UserMovieRating_CreateOrUpdate** Action in the new dialog.



- d. Back in the **ThumbsClickedHandler** Action, select the **UserMovieRating_CreateOrUpdate** to see its properties. Expand the **UserMovieRating** input parameter of the Action, just like the image indicates.



- e. Set the attributes of the UserMovieRating record with the following values

Id = GetUserMovieRatingsByUserId.List.Current.UserMovieRating.Id

UserId = GetUserId()

MovieId = MovieId

Rating = SelectedRating

IsRated = IsRated

The screenshot shows the configuration for the 'UserMovieRating_CreateOrUpdate' Run Server Action. The interface includes a title bar with an orange circle icon and the text 'UserMovieRating_CreateOrUpdate Run Server Action'. Below this, there are several configuration fields: 'Name' (UserMovieRating_CreateOrUpdate), 'Server Request Timeout' ((App default timeout)), 'Action' (UserMovieRating_CreateOrUpdate), and a list of attributes for 'UserMovieRating'. The attributes are: 'Id' (GetUserMovieRatingsByUserId.List.Current.I), 'UserId' (GetUserId()), 'MovieId' (MovieId), 'Rating' (SelectedRating), and 'IsRated' (IsRated). There is also a '(New Argument)' field at the bottom.

Note: It is possible to set the values individually for each attribute of a record, which is what we are doing in this case. So far, we had only assigned the complete record, but this is also a valid option.

For the **ID**, we get the record that was fetched from the database. If there is already a rating, we use the same Id, if there's not, then the Id will go as 0 and a new record will be created in the database.

For the **UserId**, we set it to be the user currently logged in. For the **MovieId**, we use the input parameter of the MovieDetail Screen of the same name. And for the **Rating** and **IsRated** values, we use the input parameters of the Action, which are coming from the Event.

5. Publish the app to save the latest changes.

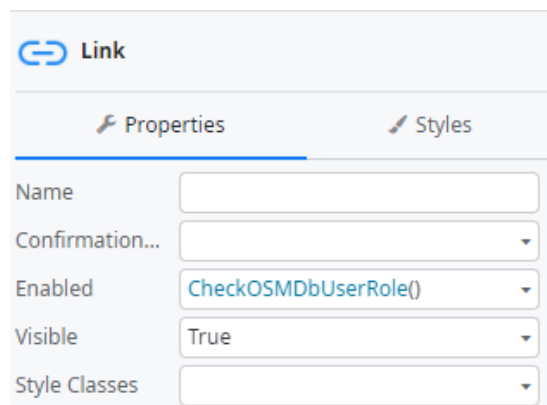


Notice that we haven't done anything with the output parameters of the Server Action, but that's fine! If you have some extra time, you can come back to this Action and implement the logic to display a success, or error message to the user, depending if the data was successfully added to the database or not.

Permissions to Rate a Movie

You surely noticed in the previous steps that the logic to save the rating in the database checks if the user has the OSMDbUser Role. This means that only users with this role will actually be able to rate a movie. So, let's also implement that behavior in the UI, by only enabling clicking the Thumbs Links if the user has the Role.

1. Set the Links in the Thumbs Block to be clickable only when the user has the OSMDbUser Role.
 - a. Double-click on the **Thumbs** Block to open it in the preview.
 - b. Select the filled thumbs up Link to see its properties. Set the **Enabled** property to *CheckOSMDbUserRole()*



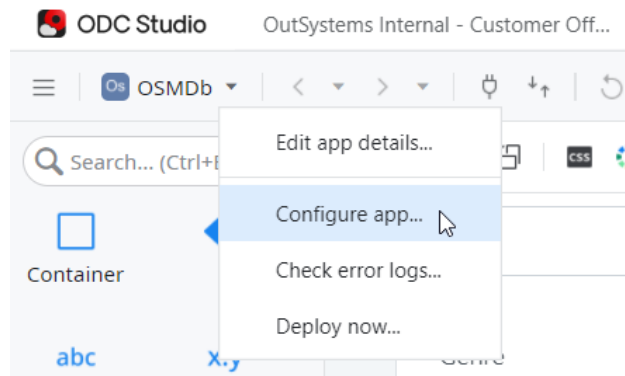
- c. Repeat the same process for all the other Links.

2. Publish the app and open it in the browser.

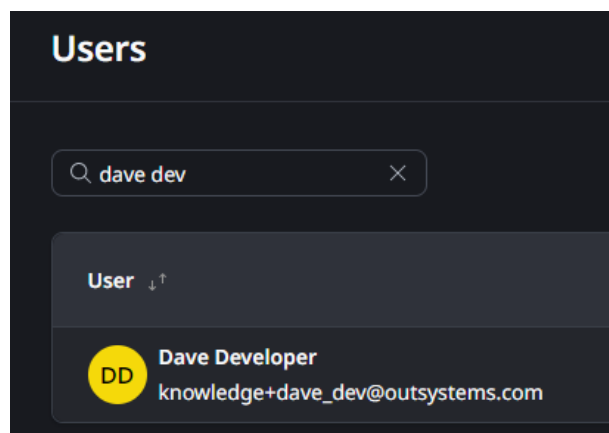


3. Try to rate a movie. Are you able to do it? You shouldn't. Why? Because you do not have this Role. Let's change that!

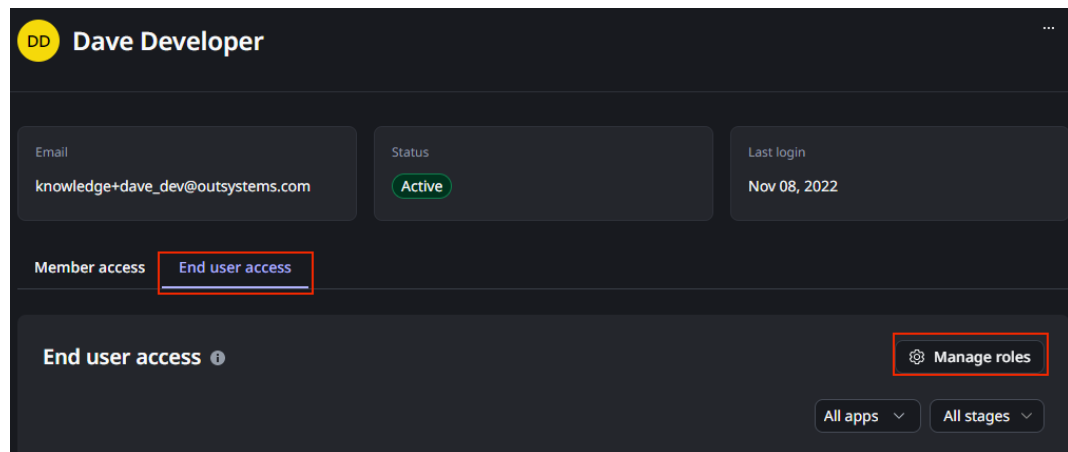
- a. In **ODC Studio**, click again in the **Configure App** option. This will open the **ODC Portal**.



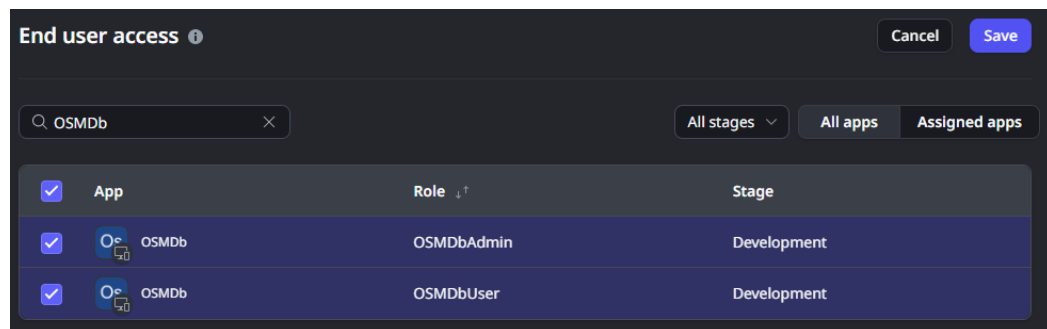
- b. In the ODC Portal, go to the **Users** section, find your user and click on it.



- c. In the user detail page, select the **End user access** tab and then click in **Manage roles**.



- d. Find your app and add the **OSMDBUser** role.



4. Now, back in the browser, log out and log in again in your OSMDB app. Try to rate a movie. Can you do it now? Well, you actually should be clicking on the thumbs, but the rating does not change right? And what if you refresh the page? It should appear now. If that's what is happening, don't worry! The rating is being saved in the database. However, there's a final tweak we need to do.

Final Touches

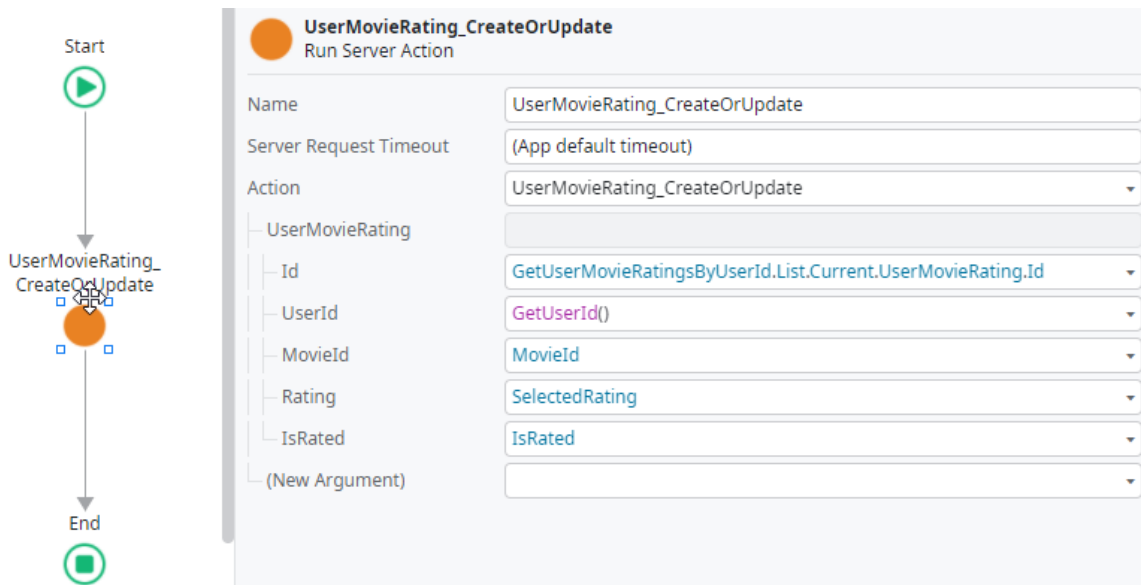
To understand what's happening, let's look at several elements that we added. First, the instance of the Block in the MovieDetail Screen. Notice the properties, where you pass the Rating and the IsRated values:

MainFlow\Thumbs

Properties Styles

Name	
Source Block	MainFlow\Thumbs
Rating	GetUserMovieRatingsByUserId.List.Current.UserMovieRating.Rating
IsRated	GetUserMovieRatingsByUserId.List.Current.UserMovieRating.IsRated
(New Argument)	

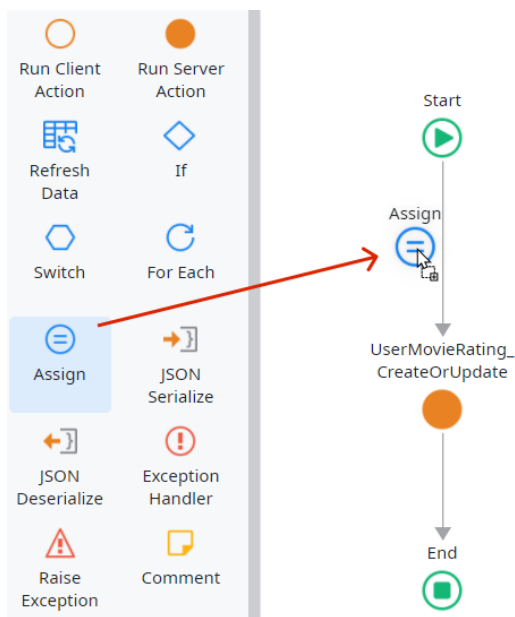
Now, let's look at the **ThumbsClickedHandler** Action and what we send to the database.



We are using the Rating and IsRated values that come from the Event, which is fine! However, how does the Block know which icons need to be displayed? The Screen is passing that information to the Block, but that information is never updated. Sure, we click on one of the icons, but when the Screen appears on the browser, it's still sending the information it fetched from the Aggregate, and not the new one. That's why when you refresh the page, the Aggregate runs again and the data is fetched, getting the result you expect. So, how to fix this?

Well, it's simple! If the values passed from the MovieDetail to the Block are what control the icons that should appear, let's update them in the **ThumbsClickedHandler** Action.

1. Open the **ThumbsClickedHandler** Action, drag an **Assign** and drop it after the Start node.



2. Define the following assignments:

GetUserMovieRatingsByUserId.List.Current.UserMovieRating.Rating = SelectedRating

GetUserMovieRatingsByUserId.List.Current.UserMovieRating.IsRated = IsRated

Well, and that's it! Right before we save the data in the database, we just update the values that are being sent to the Block. The Block gets the new information and display the respective icons that make sense with the rating being passed on. Seems strange? Well, just think that one thing is to save the data in the database and another thing is to actually update what you're seeing and using on the Screen. That's what was missing.

3. Publish the app and test it in the browser. Does the rating appear correctly now?

Awesome! And that's it!

