Lab 9: Using React state

You will continue to work on the web application you developed in the previous lab by making the filters work by means of clicks in the user interface. Also, a delete button will be added to allow deleting films. Before starting, remember to finish transforming your old application into one using React components.

1. Make the UI Filters work.

Decide where to keep the information about which filter is currently selected. Create all the suitable callbacks to implement the filters as described below. When a filter is clicked, only the films respecting the specified filter criteria must be visualized. Specifically, filtering should not affect the original list of films (regardless of where it is stored), but **only** their visualization. Of course, the information about which filter is currently selected should change and should be kept (as a state).
To implement the filtering, you must associate the items in the sidebar to the following actions:

- **All**: display *all* the films (default filter).
- **Favorite**: display only films marked as *favorite.*
- **Best Rated**: display only films whose score is *five out of five*.
- **Seen Last Month**: display only films watched *between today and the last 30 days*.
- **Unseen**: display only films *without a watch date*.

2. Allow deleting films.

Add a delete button (e.g., with the trash icon) at the end of the row for each film. When pressing the button, the film must be deleted from the list of films. Thus, the film will never appear again, regardless of the selected filter.
Remember that if you want to reload all films, at the moment you need to reload the whole application.
Decide where to keep the information about the films. Note that the film list must be modified, thus it must be kept into a state. Decide in which component it must be kept and define suitable callbacks to act on such state when deleting a film.

Hints:

1. Exploit React to update the visualization. Thus, think about which actions should modify which state and how to do it (for instance, via callbacks passed through props). React will do the rest by updating the visualization of components for which the state and/or props have changed.
2. Remember the rules to update an array used as a state: if the new state value depends on the previous one, use a callback in the setState function!