**Interactivity-JS-Lab**

**MOVIE LIST**

**Summary**

In this lab, you’ll practice more vanilla JS DOM manipulation by creating a simple Movie List project. The basic HTML and CSS have been provided for you, and you will be adding in the JavaScript to make the interface interactive. Users should be able to type in a movie title and click ‘add’ to add a movie onto the list. Clicking on the movie’s title should cross it off. Clicking the ‘x’ button next to the title should remove the movie from the list. We’ll also be making a dynamic notification.

**Setup**

Download the contents for this lab and open the starter code in VS Code.

**Step 1**

**Summary**

In this step, we’ll create our Javascript file and connect it to our HTML.

**Instructions**

* Create a new file in this folder called ***index.js***.
* Add a ***console.log(‘hello world’)*** so we can see if the script is running.
* Open ***index.html***
* Inside the ***<body>*** tag, but after the ***<main>*** element, add a ***<script>*** tag to bring in your ***index.js*** file using the ***src*** attribute.
* Open ***index.html*** in your web browser. - You can right click on the file name in VSCode and select ‘Copy Path’ then paste that path in your browser or you can open it using live-server - Open the console (right click anywhere in your browser and click ‘inspect’ or ‘inspect element’, switch to the Console tab) and look for the ***hello world*** console log from your JavaScript file. If it doesn’t appear, check the file path in your ***<script>*** tag’s ***src*** attribute.

**Step 2**

**Summary**

In this step, we’ll start creating our JavaScript function for adding a movie to our list when the ***Add*** button is pressed. We’ll need to add the event listener, get the value from the input box, and create a new element in the list.

**Instructions**

* In ***index.js***, create a new function called ***addMovie*** that takes in an ***event*** as a parameter.
* We are going to want to get and set the value of our input field in this function, so let’s select it now. Use ***querySelector*** to get the input, save it to a variable called ***inputField***.
* Let’s make the HTML for our movie list items. Create a new variable called ***movie***, store a new ***li*** element in it using ***document.createElement***, this will be the parent element of our movie’s title and the movie’s delete button.
* Next create a new ***span*** element and save it to a variable called ***movieTitle***. Set the ***textContent*** of ***movieTitle*** to be the ***value*** of ***inputField***. This will write what the user typed out into our new span.
* Now we’ll need to append the ***movieTitle*** span to our ***movie*** element. Use the ***appendChild*** method on ***movie***, passing in ***movieTitle*** to attach the title to its parent.
* Then, use ***querySelector*** to find the ***ul*** element that already exists in our HTML and use ***appendChild*** to attach the ***movie*** element we created to the list.
* Finally, outside of your function, use ***querySelector*** to select the ***form*** element and then use ***addEventListener*** to listen for a ***submit*** event on the ***form*** element and run the ***addMovie*** function.
* Save your files, run the project and give it a try – it doesn’t work the way we expected. Because our ***button***is inside a ***form*** element, it has a default action that is also running and interfering with our code. To fix this, at the beginning of the ***addMovie*** function, add ***event.preventDefault()***
* For a more user-friendly experience, let’s clear the input field when the ‘Add’ button is clicked so it’s ready for another movie. To do this, set the ***value*** of ***inputField*** to an empty string at the bottom of the ***addMovie*** function.

**Step 3**

**Summary**

Now that we can add a movie list item, we need to be able to remove them as well. In this step, we’ll make some changes to our ***addMovie*** function so we have a way to remove movies. We’ll need to add a delete button to each movie list item and create an event listener for it.

**Instructions**

* In the ***addMovie*** function, after appending the ***movieTitle*** to ***movie***, use ***createElement*** to create a new ***button*** element and save it to a variable called ***deleteBtn***.
* Set the ***textContent*** of ***deleteBtn*** to be the letter ***X***.
* Use ***addEventListener*** to listen for a ***click*** event on the button and run the ***deleteMovie*** function. We will create that function later in this step.
* Now that the button has been created, use the ***appendChild*** method to add ***deleteBtn*** onto the ***movie***element.
* Finally, outside of the ***addMovie*** function, create a new function called ***deleteMovie*** that takes in an ***event***parameter. When we click the button, we want to remove the entire list item. Since the button is a child of the list item, we can use ***event.target.parentNode.remove()*** to remove the entire list item. JavaScript knows what the target of this event is (the specific delete button that’s clicked) and will only get rid of that one button’s parent (the movie list item that holds the title and button). You should now have a functioning delete button on each movie you add!

**Step 4**

**Summary**

Now that we can add and remove movies from our list, we can add to our app by allowing users to mark items as watched by clicking on a movie’s title. The CSS has already been set up to display list items differently if they have the ***checked*** class. We need to create a function that will toggle the ***checked*** class on any movie title.

**Instructions**

* In ***index.js***, create a new function called ***crossOffMovie*** that takes in an ***event*** as a parameter. - Later, we will need to add this as an event handler for every movie title span.
* Call ***event.target.classList.toggle()*** passing in the ***checked*** class so that the class is added or removed if the title is clicked. (We want users to be able to “un-cross” movies off in case they did it by accident).
* Finally, we need to add this function as an event handler for every movie title. In the ***addMovie*** function, after you set the ***textContent*** of the ***span*** element, use ***addEventListener*** to listen for a ***click*** event on the span and run the ***crossOffMovie*** function.

**Step 5**

**Summary**

Finally, let’s add in some functionality that will display messages to users to let them know what action took place. We’ll be selecting an existing, but blank, HTML element and adding some code to our ***deleteMovie*** and ***crossOffMovie*** functions.

**Instructions**

* At the top of your JS file, select the HTML element with the ***message*** id using ***querySelector***. Save it to a variable called ***message***. Right now, it’s just an empty ***aside*** element. In the next steps, we’ll assign its ***textContent*** to send users different notifications.
* In the ***deleteMovie*** function, add a line that assigns the ***textContent*** of ***message*** to be a string that says something like ‘Movie deleted!’ - You should now see this message pop up when you delete a movie. Test it out!
* In the ***crossOffMovie*** function, we’re going to do something similar, but we want to have different messages based off of whether the movie was just checked off as ‘watched’ or if it was added back to the list. So let’s start by creating the structure for an if/else block. Put it below where you toggled the ***checked***class in the ***crossOffMovie*** function.
* The condition of your ***if*** statement should check if it’s true that the ***event.target.classList*** contains the ***checked*** class. - ***contains*** is a built-in method that can be used on ***classList***, the structure for doing so is ***event.target.classList.contains(‘some-class-name’)***
* If it’s true, then change ***message***’s ***textContent*** to be a string that says something like ‘Movie watched!’
* Else, change ***message***’s ***textContent*** to be a string that says something like ‘Movie added back!’
* Test it all out!

**Core Lab Completed!**

You’ve completed the core lab! Push your code to GitHub and move on to the intermediate portion.

**INTERMEDIATE INSTRUCTIONS**

In this section, we are going to customize our notifications a bit. We’ll make it so they disappear after a set amount of time. And we’ll change the ***textContent*** to include the title of the movie that was affected.

**Step 1**

**Summary**

Let’s start by creating a function that we’ll be able to call from both ***deleteMovie*** and ***crossOffMovie*** that will hide the message after a set amount of time. By creating one function, we’ll save ourselves from writing the same code in both functions. This way, we only have to write it once and then we can call it in multiple places.

**Instructions**

* Create a function called ***revealMessage***. Inside the function, call ***setTimeout***, passing in a callback function and a time in milliseconds. - The callback function should add the ***hide*** class to ***message***, you can see what the ***hide*** class does in the CSS file - We want the callback function to run 1 second after ***setTimeout*** is invoked, so for the second argument, pass in the number 1000
* At the bottom of the ***deleteMovie*** function, call ***revealMessage***
* Invoke ***revealMessage*** at the bottom of the ***crossOffMovie*** function
* Test out your code – your notification message should disappear after the amount of milliseconds you prescribed. But it only works the first time! No new messages are showing up for subsequent actions. Let’s fix that.
* At the top of the ***revealMessage*** function, remove the ***hide*** class from ***message*** – this will ensure that the message isn’t hidden when the function is first called.

**Step 2**

**Summary**

Now, you’ll change your message’s text in the ***crossOffMovie*** function so that it tells the user which movie they just crossed off or added back in.

**Instructions**

* In the ***if*** block inside of the ***crossOffMovie*** function, change the ***message***’s ***textContent*** to be a string that says something like ‘MOVIE watched!’ where ***MOVIE*** is the title of the movie they clicked on. You can access that title in the ***textContent*** property of ***event.target***, and you can use concatenation or a template string.
* In the ***else*** block, change the string in the same way so that it contains the title of the movie
* Test it out!

**Step 3**

**Summary**

Last, we’ll add the movie title into the notifications message from ***deleteMovie***.

**Instructions**

* In the ***deleteMovie*** function, change the string you’re assigning to the ***message***’s ***textContent*** to contain the deleted movie’s title. - This will be different than how we accessed the titles in the ***crossOffMovie***function. Previously, we were accessing the ***textContent*** of the element that we clicked. But the ***deleteMovie*** function runs when the ‘X’ button is clicked. So the title is actually a sibling element of the button. You’ll need to explore the properties available on ***event.target.parentNode*** to get at that movie title text.

**Wrapping Up**

**Submit to GitHub**

Don’t forget to push your code to github.