JavaScript Challenge 2 - Movie Search Engine With the OMDb API

Introduction:

The goal of this exercise is to create a movie search engine using the OMDb API. This API provides detailed information about movies, including the title, plot, poster image, ratings, and more. You will be working with asynchronous JavaScript, making API calls, handling JSON data, and dynamically manipulating the DOM. The HTML and CSS will be provided, and your task is to write the JavaScript code.

Instructions:

Step 1: Define Constants for OMDB API URL

Store the URL for the OMDB API in an `OMDB_URL` constant. This API provides detailed information about movies.

Step 2: Grab References to HTML Elements

Reference specific HTML elements in your JavaScript code to handle user input and update the content.

- `searchButton` : Reference to the button for searching movies.
- `searchInput` : Reference to the input field for movie titles.
- `movieContainer` : Reference to the container where movie details will be displayed.

Step 3: Attach Event Listener for Search Button

Add a click event listener to the search button, which will call the `searchMovie` function and clear the search input value by setting it to an empty string.

Step 4: Define an Asynchronous Function `searchMovie`

Declare an asynchronous function that takes a movie title as an argument and fetches the movie details from the OMDB API.

```
async function searchMovie(title) {
  // Fetching and handling code
}
```

Inside `searchMovie`:

1. **Fetch Data**: Utilize `fetch` to request movie details from the OMDB API. Await the response and parse it to JSON.

2. **Create HTML Template**: Define a string and name it `template`. Use this template literal to create HTML code (you will find the desired HTML template in the `movie.html` file in this exercise folder) representing the movie's information, including title, poster, plot, genre, year, director, actors, and ratings.

3. **Insert HTML into Container**: Assign the HTML template to the `innerHTML` property of `movieContainer`, displaying the movie details on the page.

4. **Handle Ratings**: Call the `handleRatings` function with the fetched ratings and the target div for displaying them.

5. **Error Handling**: Use a try-catch block to handle any potential errors during fetching, and log the error to the console.

Step 5: Define Function `handleRatings`

This function takes the ratings and a target div as parameters and appends the ratings to the target div. It filters the ratings based on specific sources and creates the required HTML elements.

1. **Function Definition**:

- Declare a function named `handleRatings` that takes two parameters: `ratings` (an array of rating objects) and `targetDiv` (a reference to a DOM element where the ratings will be appended).

2. **Check for Ratings**:

- Inside the function, use an `if` statement to check whether the `ratings` parameter is provided.
- If `ratings` is `null` or `undefined`, the function should not proceed further.

3. **Iterate Through Ratings**:

- Use the `forEach` method to iterate through each rating object in the `ratings` array.
- Inside the `forEach` callback, the current rating object is accessed using a `rating` constant.

4. **Filter Specific Rating Sources**:

- Check if the `rating.Source` is one of **"Internet Movie Database"**, **"Rotten Tomatoes"**, or **"Metacritic"** using the `includes` method. If the condition is met, proceed to create the elements; otherwise, skip to the next iteration.

```
  if (["Internet Movie Database", "Rotten Tomatoes",
"Metacritic"].includes(rating.Source))
```

5. **Create Rating Div**:

- Create a new `div` element and store it in a constant named `ratingDiv`.
- This div will contain the source and value of the rating.

6. **Create Source Span**:

- Create a new `span` element and store it in a constant named `sourceSpan`.

7. **Create Value Span**:

- Create a new `span` element and store it in a constant named `valueSpan`.

8. **Style the `sourceSpan` Span**:

- Set the `fontWeight` style property of the `sourceSpan` to 'bold' to highlight the source name.

9. **Set the inner text of the spans**

- Use a template literal to set the `innerText` property to the current `rating.Source` value followed by a colon and a space.

```
  sourceSpan.innerText = `${rating.Source}: `;
```

- Set the `innerText` property to the current `rating.Value`.

10. **Append Spans to Rating Div**:

- Append both the `sourceSpan` and `valueSpan` to the `ratingDiv` using the `appendChild` method.

11. **Append Rating Div to Target Div**:

- Finally, append the `ratingDiv` to the `targetDiv` using the `appendChild` method.

```
ratingDiv.appendChild(sourceSpan);
ratingDiv.appendChild(valueSpan);
```

12. **End of Function**: - The function end with the appending of the ratings div to the `targetDiv`. - The function does not need to return a value.

Step 6: Initial Movie Search

Call the `searchMovie` function with a predefined movie title to search for and display details when the page is loaded.

Extra Challenge

Extend the functionality of your movie search engine by taking on the following additional challenges:

1. **Enhanced Error Handling**: If the user performs a search with an empty input, display an error message either as an alert or as a customized error message inside the UI.

Instead of simply logging an error to the console, display a friendly error message in the UI, perhaps inside the `movieContainer`. This will help in informing the user if something goes wrong. You can consider creating a function to handle this.

4. **Enter Key Search Support**: Allow users to initiate a search by hitting the Enter key while focusing on the search input. You can add a key event listener on the `searchInput` and look for the Enter key code.

5. **Loading Indicator**: Implement a loading indicator that shows up while the API request is in progress and hides once the response is received. This gives users feedback that the system is working on their request.

6. **Rating Source Customization**: Provide checkboxes or a dropdown to allow users to select which rating sources they want to see. Modify the `handleRatings` function accordingly to filter based on the user's selection.