

Exercise: Building a Movie Rating List

Objective:

Create a web page that fetches movie details from an external API and displays them using the provided HTML and CSS files.

Perquisites

In this exercise, you will work with this API: `themoviedb`. Please register to it. You will find the relevant documentation for this exercise in this [link](#). There is a `data.json` in this exercise folder that shows the data structure that is being received from this API.

Steps:

1. Define a Movie Class:

Create a class named `Movie` with the following properties:

- `title`
- `releaseDate`
- `picture`
- `rating`

Add the following methods to the class:

- `getTitle` : Returns the title
- `getReleaseDate` : Converts the release date to a localized date string and returns it
- `getPicture` : Returns the picture URL
- `getRating` : Returns the rating

```
class Movie {
  constructor(title, releaseDate, picture, rating) {
    this.title = title;
    this.releaseDate = releaseDate;
    this.picture = picture;
    this.rating = rating;
  }
  // Define getTitle, getReleaseDate, getPicture, getRating methods
  here...
}
```

2. Create a function to get the Headers and Authorization:

Create a `getHttpOptions` function that will return an object with the `method` set to 'GET' and include headers with the `accept` field set to 'application/json'. Include the Authorization header with the provided [Bearer token](#).

```
const getHttpOptions = () => ({
  method: 'GET',
  headers: {
    accept: 'application/json',
    Authorization: 'Bearer your-bearer-token-here'
  }
});
```

3. Set Up API Constants:

- Define in a constant the base URL for the movie database API

```
const API_BASE_URL =
  'https://api.themoviedb.org/3/';
```

3. Create a function to get a movie image URL:

- Write a function `getImage` to return the image URL from the image path, accepting a size parameter:
 1. **Define the Function Name and Parameters:** Begin by defining a constant named `getImage`. This function will take one parameter: `imgPath`, which is the specific path to the image.

2. **Utilize Template Literals:** Inside the function, you will return a template literal.

3. **Construct the URL:** Begin the URL with the base path for images:

```
https://image.tmdb.org/t/p/original/ .
```

4. **Append the Image Path Parameter:** After the forward slash `/` add the `imgPath` parameter using `${imgPath}`. This includes the specific image path in the URL.

4. Write a Function to Fetch Movies:

- **Create a Function Named `fetchMovies`:** Define the `fetchMovies` function to handle fetching movie data from the now-playing endpoint.
- **Use the `fetch` API for the GET Request:** Utilize the `fetch` function to make a GET request to the now-playing endpoint by combining the API base URL with the specific path and also pass the `getHttpOptions()` function, to pass the necessary options with the request.

```
fetch(`${API_BASE_URL}movie/popular?language=en-US&page=1`,  
getHttpOptions())
```

- **Handle the Response:** Use `.then` to transform the response to JSON.

```
.then(response => response.json())
```

- **Iterate Through the Results and Create Movie Instances:** In the next `.then` block, iterate through the `results` of the transformed response using `.forEach`. For each item, create a new instance of the `Movie` class with the relevant properties (title, release date, picture, and rating). Utilize the provided `getImage` function for the image path. Pass to the `getImage` function the `item.poster_path` **or** the `item.backdrop_path`, since the item may come from the API without `item.poster_path` property. After creating a new instance of the `Movie` class, call the `createMovieCard()` function (we will write it later).

```

fetch(`${API_BASE_URL}movie/now_playing?language=en-US&page=1`,
options)
  .then(response => response.json())
  .then(response => {
    response.results.forEach(item => {
      // Your code here...
    })
  })
  .catch(err => console.error(err));

```

- **Update the Item Count:** Call the `updateItemCount` function after iterating through the movies.
- **Handle Errors:** Implement a `.catch` block to log any errors that may occur.

4. Create a Function to Update Item Count:

1. **Define the Function:** Start by defining a function called `updateItemCount` that takes no parameters.
2. **Select the Items Count Element:** Within the function, you will need to select the paragraph element where the count of items will be displayed. Use the `querySelector` method to select the paragraph inside the element with a class of "sort-bar."
3. **Select All Movie Elements:** Next, you need to find out how many movie elements are in the document. Use the `querySelectorAll` method to select all the elements with a class of `movie`. This will give you a `NodeList` of movie elements.
4. **Calculate the Number of Movies:** Determine the number of movie elements by accessing the `length` property of the `NodeList` obtained in the previous step.
5. **Update the Text Content:** Finally, update the text content of the paragraph element selected earlier with the number of movies. Use a template literal to format the text, incorporating the number of movies followed by the word "items."

5. Write a Function to Create Movie Cards:

Create the `createMovieCard` function that builds the HTML for each movie. Use the HTML code of the `movie.html` file in this exercise files.

Don't forget to insert real data in the relevant places like in this example, such as the `src` attribute and other movie details.

```
function createMovieCard(movie) {
  let target = document.querySelector(
    '.movies-list',
  );
  target.innerHTML += `
    <div class="movie">
      
      <a href="#">
        
      </a>
      <img class="poster" src=${movie.getPicture()}>
      <!-- Add other movie details here... -->
    `;
}
```

6. Call the fetchMovies Function:

Finally, call the `fetchMovies` function to start fetching movies:

```
fetchMovies();
```

Tips:

- Remember to handle errors with a `.catch` block in the fetch function
- Make sure to utilize the Movie class methods within the `createMovieCard` and `fetchMovies` functions
- Test each part of the code as you build it to ensure everything is working correctly

Challenge:

- Extend the functionality by showing a special error message in the HTML if there is any error. Write a function `createErrorMessage` that will be similar to the `createMovieCard` function, to display an error message, in case of an error.

Conclusion:

This exercise allows you to apply concepts of classes, API calls, and DOM manipulation to build a functional web application. By following the steps and code snippets, you will create a movie

list web page that connects various JavaScript concepts you have learned in the course up til now.