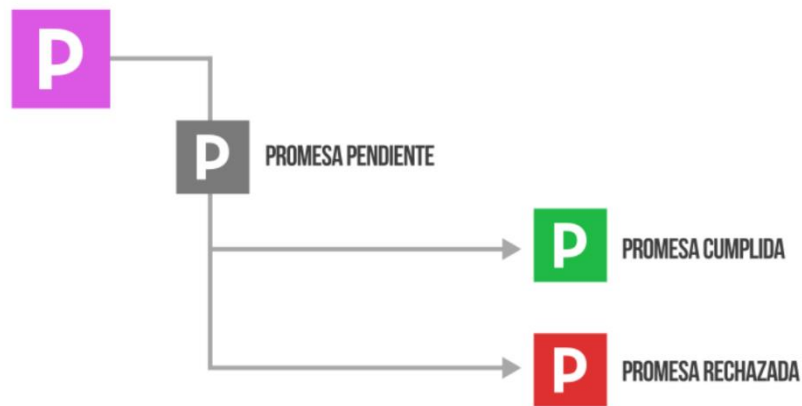


Promesas

Una Promise es un objeto que representa la terminación o el fracaso de una operación asíncrona. Esencialmente, una promesa es un objeto devuelto al cuál se adjuntan funciones callback.



Las promesas en Javascript se representan a través de un objeto, y cada promesa estará en un estado concreto: pendiente, aceptada o rechazada. Además, cada promesa tiene los siguientes métodos, que podremos utilizar para utilizarla:

Métodos	Descripción
<code>.then(<small>FUNCTION</small> resolve)</code>	Ejecuta la función callback <code>resolve</code> cuando la promesa se cumple.
<code>.catch(<small>FUNCTION</small> reject)</code>	Ejecuta la función callback <code>reject</code> cuando la promesa se rechaza.
<code>.then(<small>FUNCTION</small> resolve, <small>FUNCTION</small> reject)</code>	Método equivalente a las dos anteriores en el mismo <code>.then()</code> .
<code>.finally(<small>FUNCTION</small> end)</code>	Ejecuta la función callback <code>end</code> tanto si se cumple como si se rechaza.

```
fetch("/robots.txt")
  .then(response => response.text())
  .then(data => console.log(data))
  .finally(() =>
    console.log("Terminado. "))
  .catch(error =>
    console.error(data));
```

Obsérvese además que hemos añadido el método `.finally()` para añadir una función callback que se ejecutará **tanto si la promesa se cumple o se rechaza**, lo que nos ahorrará tener que repetir la función en el `.then()` como en el `.catch()`.

FLUJO DE UNA PROMESA

Cuando una promesa es disparada esta ingresa a un estado la cual puede ser las siguientes:

fulfilled: se cumplió con éxito.

rejected: no se pudo cumplir a causa de un error. Depende de ustedes como hagan el reject.

pending: se esta procesando.

settled: simplemente finalizo.

Ejercicio #1:

```
script > JS EjercicioUno.js > ...
1
2 let x = 10;
3 const promesa = new Promise((resolve, reject) =>{
4     if(x==10){
5         resolve('La variable es igual a 10')
6     }else{
7         reject('La variable no es igual a 10')
8     }
9 });
10
11 promesa.then(res =>{
12     console.log('success' + res)
13 })
14 .catch(error =>{
15     console.log('Error' + error)
16 })
```

Ejercicio #2

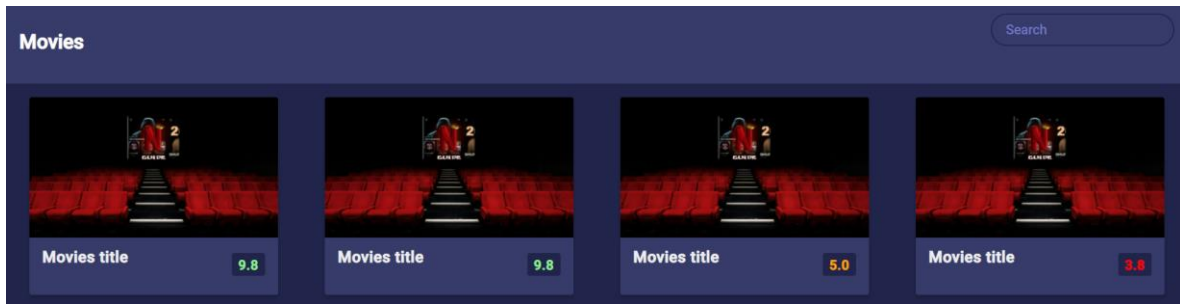
```
/* Implementación con promesas */
const doTask = (iterations) => new
Promise((resolve, reject) => {
    const numbers = [];
    for (let i = 0; i < iterations; i++)
    {
        const number = 1 +
Math.floor(Math.random() * 6);
        numbers.push(number);
        if (number === 6) {
            reject({
                error: true,
                message: "Se ha sacado un 6"
            });
        }
    }
    resolve({
        error: false,
        value: numbers
    });
});
```

Métodos	Descripción
<code>Promise.all(<small>ARRAY</small> list)</code>	Acepta sólo si todas las promesas del <small>ARRAY</small> se cumplen.
<code>Promise.allSettled(<small>ARRAY</small> list)</code>	Acepta sólo si todas las promesas del <small>ARRAY</small> se cumplen o rechazan.
<code>Promise.any(<small>OBJECT</small> value)</code>	Acepta con el valor de la primera promesa del <small>ARRAY</small> que se cumpla.
<code>Promise.race(<small>OBJECT</small> value)</code>	Acepta o rechaza dependiendo de la primera promesa que se procese.
<code>Promise.resolve(<small>OBJECT</small> value)</code>	Devuelve un valor envuelto en una promesa que se cumple directamente.
<code>Promise.reject(<small>OBJECT</small> value)</code>	Devuelve un valor envuelto en una promesa que se rechaza directamente.

Ejercicios:

- 1) Crear un algoritmo que lea el nombre de un usuario y lo imprima.
- 2) Crear un algoritmo que le pida al usuario 3 números, calcular la suma de los 3 números e imprimir el resultado si es mayor que 25
- 3) Crear un algoritmo que permita leer la edad y peso de una persona y posteriormente imprimirla, si el peso es superior a 100 imprimir que el usuario tiene sobre peso

Movies App



Crear la estructura básica de html

Vincular los archivos externos (script, css)

```
<> index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style.css">
8      <title>Movie app</title>
9  </head>
10 <body>
11     <h1>Movie app</h1>
12     <script src="style/style.css"></script>
13 </body>
14 </html>
```

Agregar la cabecera de la página con un buscador

```
<header>
  <form id="form">
    <input
      type="text"
      id="search"
      class="search"
      placeholder="Search"
    />
  </form>
</header>
```



Para el contenido principal de la aplicación vamos a crear un selector de tipo clase con el nombre movie, agregar una imagen

```
<main id="main">
  <div class="movie">
    
  </div>
</main>
```

Agregar información referente a la película

```

<main id="main">
  <div class="movie">
    
    <div class="movie-info">
      <h3>Movies title</h3>
      <span class="green">9.8</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Assumenda quas vero neque sint quisquam sit. Exercitationem sint temporibus dicta?
    </div>
  </div>
</main>

```

Agregar estilos básicos:

Fuente: Roboto

```

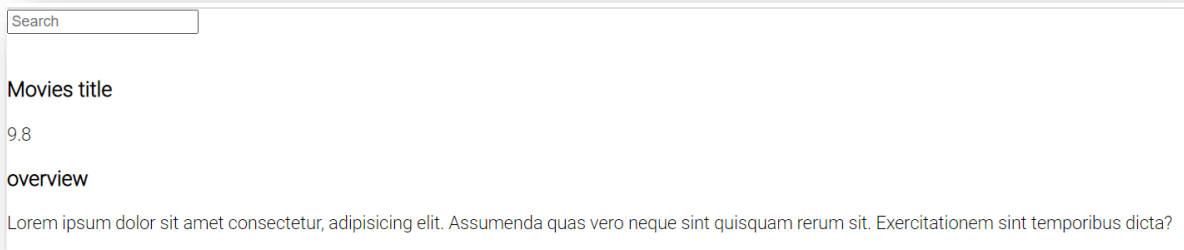
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swap');

*{
  box-sizing: border-box;
}

body{
  font-family: 'Roboto', sans-serif;
  margin: 0;
}

```

Resultado:



Crear variables en el archivo css

```

:root{
  --primary-color: #22254b;
  --secondary-color: #373b69;
}

```

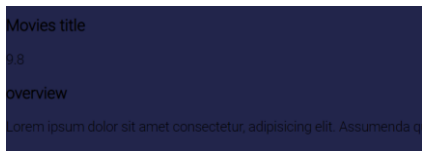
Usar variable

```

body{
  background-color: var(--primary-color);
  font-family: 'Roboto', sans-serif;
  margin: 0;
}

```

Resultado:



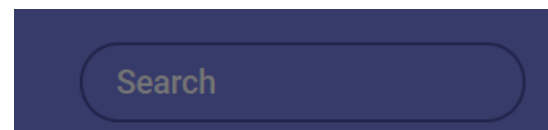
Agregar estilos a la etiqueta search



```
18 header{
19   padding: 1rem;
20   display: flex;
21   justify-content: flex-end;
22 }
23 |
24 .search{
25   background-color: transparent;
26   border: 2px solid var(--primary-color)
27 }
```

Agregar más estilos:

```
.search{
  background-color: transparent;
  border: 2px solid var(--primary-color);
  border-radius: 50px;
  font-family: inherit;
  font-size: 1rem;
  padding: 0.5rem 1rem;
  color: #fff;
}
```



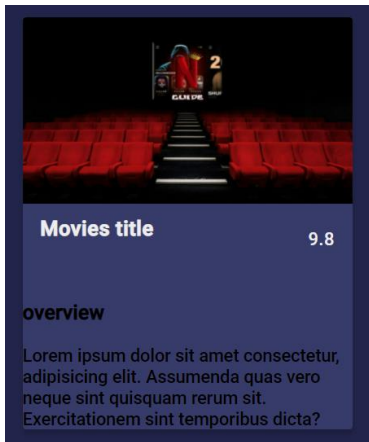
Agregar pseudo elementos y pseudoclases

```
.search::placeholder{
  color: #7378c5;
}

.search:focus{
  outline: none;
  background-color: var(--primary-color);
}
```



Darle estilos a la estructura de la película



```
✓ .movie{
  width: 300px;
  margin: 1rem;
  background-color: var(--secondary-color);
  box-shadow: 0 4px 5px 0 rgba(0, 0, 0, 0.2);
  position: relative;
  overflow: hidden;
  border-radius: 3px;
}

✓ .movie-info{
  color: #eee;
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0.5rem 1rem 1rem;
  letter-spacing: 0.5px;
}

✓ .movie-info h3{
  margin-top: 0;
}

✓ .movie img{
  width: 100%;
}
```

Agregar estilos de la etiqueta span

```
.movie-info span{
  background-color: var(--primary-color);
  padding: 0.25rem 0.5rem;
  border-radius: 3px;
  font-weight: bold;
}
```

9.8

Duplicar la estructura del html para las diferentes calificaciones de categorías de películas
cada película tiene una calificación la cual la vamos catalogar en green, orange, red

```
<main id="main">
  <div class="movie">
    
    <div class="movie-info">
      <h3>Movies title</h3>
      <span class="green">9.8</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Assumenda quas vero neque sint quisquam rerum sit. Exercitationem sint temporibus dicta?
    </div>
  </div>

  <div class="movie">
    
    <div class="movie-info">
      <h3>Movies title</h3>
      <span class="orange">5.0</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Assumenda quas vero neque sint quisquam rerum sit. Exercitationem sint temporibus dicta?
    </div>
  </div>

  <div class="movie">
    
    <div class="movie-info">
      <h3>Movies title</h3>
      <span class="red">3.8</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Assumenda quas vero neque sint quisquam rerum sit. Exercitationem sint temporibus dicta?
    </div>
  </div>
</main>
```

Agregar estilos según el selector tipo clase del span


```

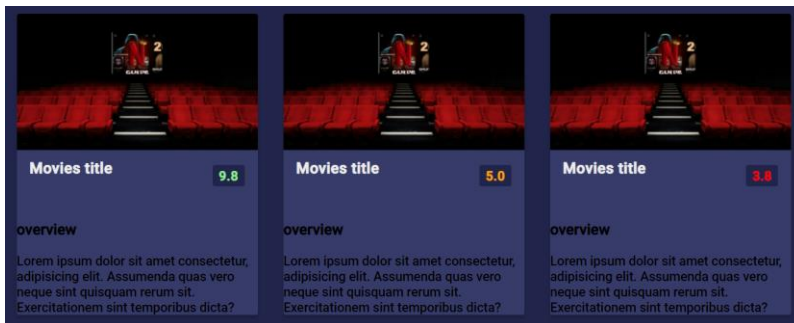
.movie-info span.green{
  color: lightgreen;
}

.movie-info span.orange{
  color: orange;
}

.movie-info span.red{
  color: red;
}

```

Resultado:

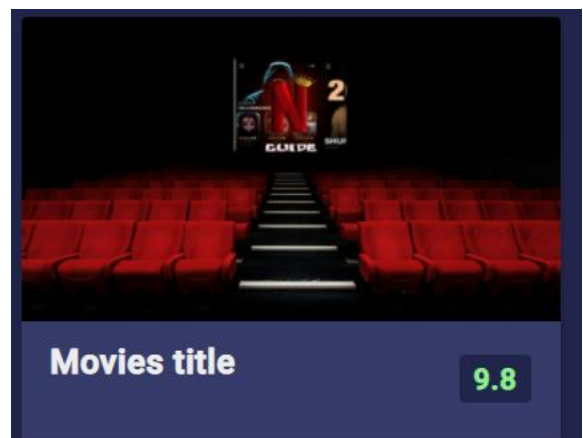


Darle estilos a la informacion de la pelicula

```

.overview{
  background-color: #fff;
  padding: 2rem;
  position: absolute;
  left:0;
  bottom: 0;
  right: 0;
  max-height: 100%;
  transform: translateY(101%);
}

```



Agregar una transformacion para observar el contenido de la pelicula

```

.movie:hover .overview{
  transform: translateY(0);
}

```

Con la propiedad hover se desplaza el contenido de la pelicula



Agregamos una transition para desplazar el contenido en el eje y

```
.overview{
  background-color: #fff;
  padding: 2rem;
  position: absolute;
  left:0;
  bottom: 0;
  right: 0;
  max-height: 100%;
  transform: translateY(101%);
  transition: transform 0.3s ease-in;
}
```

Ejercicios:

1. Leer y profundizar las tematicas que no tenga claro o le de dificultad implementarlo
2. Agrear el titulo de **Movie** en el header
3. Agregar el mismo espacio entre todos los componentes (usar propiedades css de flex)
4. Revisar el contenido del Themoviesdb

Themoviesdb:

<https://www.themoviedb.org/?language=es>

<https://developers.themoviedb.org/3/getting-started/introduction>



En el archivo script.js

```
const API_URL = 'https://api.themoviedb.org/3/discover/movie?sort_by=popularity.desc&api_key=3fd2be6f0c70a2a598f084ddfb75487c&page=1'
const IMG_PATH = 'https://image.tmdb.org/t/p/w1280'
const SEARCH_URL = 'http://api.themoviedb.org/3/search/movie?api_key=3fd2be6f0c70a2a598f084ddfb75487c&query='
```

Crear variables con los elementos de HTML necesarios

```
const form = document.getElementById('form')
const search = document.getElementById('search')
const main = document.getElementById('main')
```

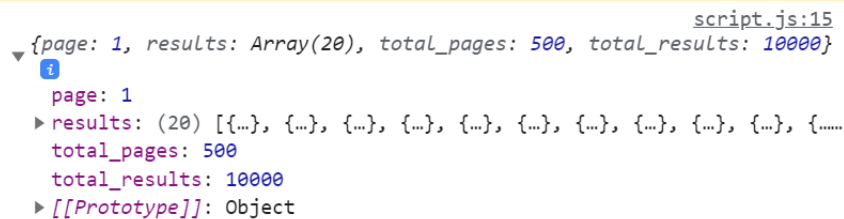
Consumir Promesas:

Crear una función para consumir el api, todo este proceso es de forma sincrónico

```
async function getMovies(url) {
  const res = await fetch(url)
  const data = await res.json()
  console.log(data.results)
  showMovies(data.results)
}

getMovies(API_URL)
```

En la función para obtener las películas invocamos una función de mostrar películas, el cual necesita parámetro para poder ser ejecutada. En la función showMovies mandamos la información que está llegando desde la url del API.



```
script.js:15
{page: 1, results: Array(20), total_pages: 500, total_results: 10000}
  page: 1
  results: (20) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}]
  total_pages: 500
  total_results: 10000
  [[Prototype]]: Object
```

El resultado de la data que llega por la solicitud al servidor de themoviedb, nos permite observar que los datos vienen internamente en un objeto llamado results.

Para consumir los datos del objeto results debemos navegar por niveles por medio del signo punto(.) como lo vemos en la imagen.

Otra forma de obtener el mismo resultado es usando **.then** y obtenemos el mismo resultado

```
const getMovies = (url) => {
  const petition = fetch(url)
  petition.then(res=>{
    res.json().then(data =>{
      console.log(data)
    })
  })
}
```

Como ven en la imagen se logra la data es perada, pero este código no esta limpio, no es una forma estándar de trabajar.

```
const getMovies = (url) => {
  const petition = fetch(url)
  petition.then(res=> res.json())
    .then(data => showMovies(data.results))
}
```

La solución a la problemática anterior, la resolvemos con peticiones anidadas como se ve en la imagen.

Si la información sigue estando objetos mas internos, seguimos utilizando el punto hasta el nivel que necesitemos.

```
showMovies(data.results)
```

Eliminar la estructura de la película en el index.html, para agregar todo este contenido desde el archivo script.js

```

<main id="main">
  <div class="movie">
    
    <div class="movie-info">
      <h3>Movies title</h3>
      <span class="green">9.8</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Assumenda qua
      sit. Exercitationem sint temporibus dicta?
    </div>
  </div>
</main>

```

Borrar el contenido del interior de la etiqueta main

```

<body>
  <header>
    <h2>Movies</h2>
    <form id="form">
      <input type="text" id="search" class="search" placeholder="Search" />
    </form>
  </header>
  <main id="main"></main>
  <script src="./script/script.js"></script>
</body>

```

Agregar la estructura de la película desde el archivo de JavaScript

1. Limpiar el contenido de la etiqueta main

```
main.innerHTML = ''
```

2. Recorrer el array de películas

```

movies.forEach((movie) => {
  });

```

3. Cada vez que se recorra la información de la película vamos a crear una película con la estructura que habíamos realizado en html

4. Para consumir la información debemos desestructurar los datos de las películas

```
const { title, poster_path, vote_average, overview } = movie
```

5. Crear una etiqueta para ser insertada en el DOM

6. Agregar una clase movie para darle el formato que se asigno para esta sesión en el archivo css

```

const movieEl = document.createElement('div')
movieEl.classList.add('movie')

```

7. En el contenido del <div> que creamos vamos a enviar la estructura de la película

```

movieEl.innerHTML = `
  
  <div class="movie-info">
    <h3>${title}</h3>
    <span class="${getClassByRate(vote_average)}">${vote_average}</span>
  </div>
  <div class="overview">
    <h3>overview</h3>
    ${overview}
  </div>
`

```

Para finalizar debemos agregar la etiqueta al archivo HTML

```
main.appendChild(movieEl)
```

Organizar el contenido para consumir la información de las películas

La estructura completa para obtener la información de las películas

```

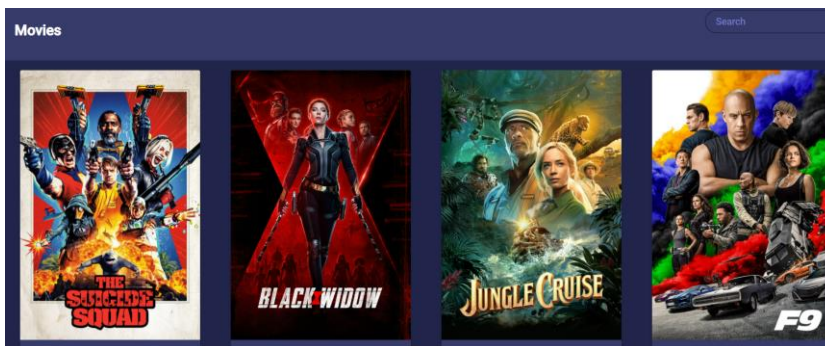
movies.forEach((movie) => {
  const { title, poster_path, vote_average, overview } = movie

  const movieEl = document.createElement('div')
  movieEl.classList.add('movie')
  movieEl.innerHTML = `
    
    <div class="movie-info">
      <h3>${title}</h3>
      <span class="${getClassByRate(vote_average)}">${vote_average}</span>
    </div>
    <div class="overview">
      <h3>overview</h3>
      ${overview}
    </div>
  `

  main.appendChild(movieEl)
});

```

Resultado en el navegador



Para darle funcionalidad al buscador, debemos agregar una función para escuchar cuando se ejecute el evento de submit.

Para evitar que la aplicación se ejecute de nuevo, vamos a prevenir que ese evento se ejecute

```

form.addEventListener('submit', (e) => {
  e.preventDefault()
})

```

Capturamos la información de ese input de búsqueda

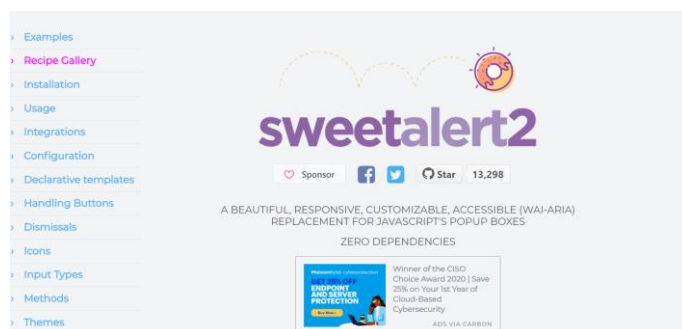
```
const searchTerm = search.value
```

Validamos si tenemos información para hacer la búsqueda, de lo contrario se vuelve a recargar la pagina

```
form.addEventListener('submit', (e) => {  
  e.preventDefault()  
  const searchTerm = search.value  
  if (searchTerm && searchTerm !== '') {  
    getMovies(SEARCH_URL + searchTerm)  
    search.value = ''  
  } else {  
    window.location.reload()  
  }  
})
```

Realizar las validaciones respectivas en la App

Instalar sweetalert a la aplicación, ir a la siguiente página <https://sweetalert2.github.io/#download>



La instalación la vamos a realizar por medio del CDN

```
<script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script>
```

Agregar esta etiqueta al archivo HTML

```

<body>
  <header>
    <h2>Movies</h2>
    <form id="form">
      <input type="text" id="search" class="search" placeholder="Search" />
    </form>
  </header>
  <main id="main"></main>
  <script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script>
  <script src="./script/script.js"></script>
</body>

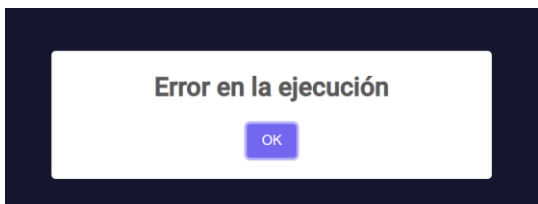
```

En la función donde obtenemos la data, es conveniente validar si la información se genera correctamente, si surge un error se activa una alerta de la librería de sweetalert

```

async function getMovies(url) {
  try {
    const res = await fetch(url)
    const data = await res.json()
    showMovies(data.results)
  } catch (error) {
    Swal.fire('Error en la ejecución',error)
  }
}

```



Ejercicio:

- 1) Consumir el API de Rick and Morty e imprimirlas como se ve en la imagen.
<https://rickandmortyapi.com/api/character>

The Rick and Morty API

