Studio Ghibli Part 1

Introduction to 3rd Party APIs

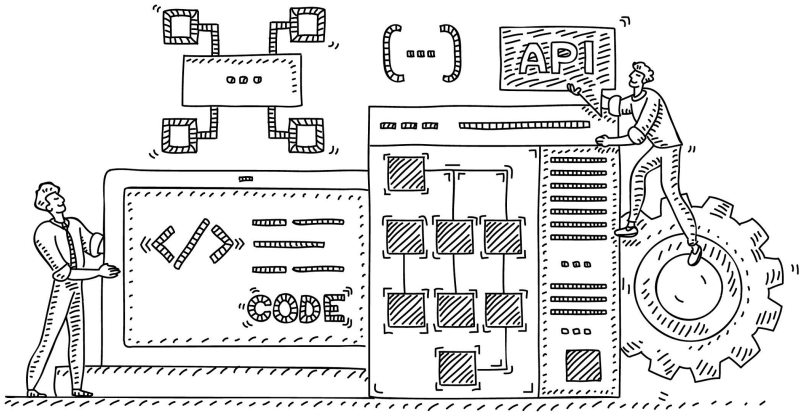
What is an API?

- An application interface provides access to features of a program or function.
- Just like how the accelerator in a car performs functions under the hood that you don't need to know about.

You're Already Working with APIs

We have been using APIs all throughout our lessons, without specifically calling attention to them.

Fetch and setTimeOut are browser APIs. So is accessing the DOM through methods like getElementById.



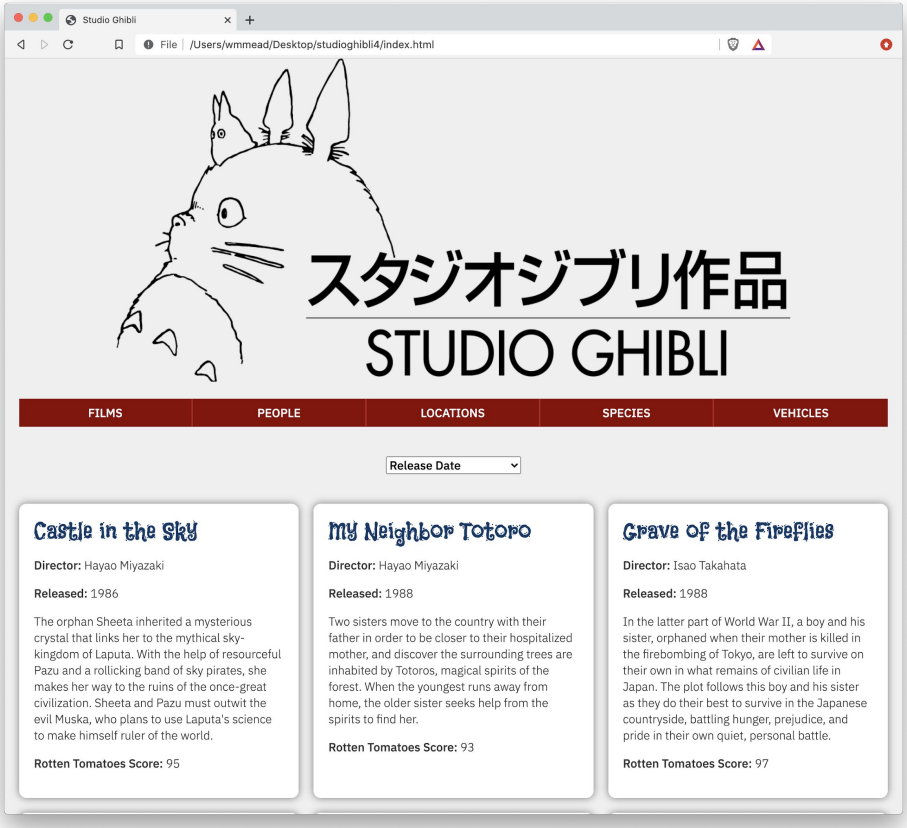
APPLICATION PROGRAMMING INTERFACE

3rd Party APIs

3rd party APIs frequently provide data you can interact with, usually in a JSON format, which is easily consumed by JavaScript.

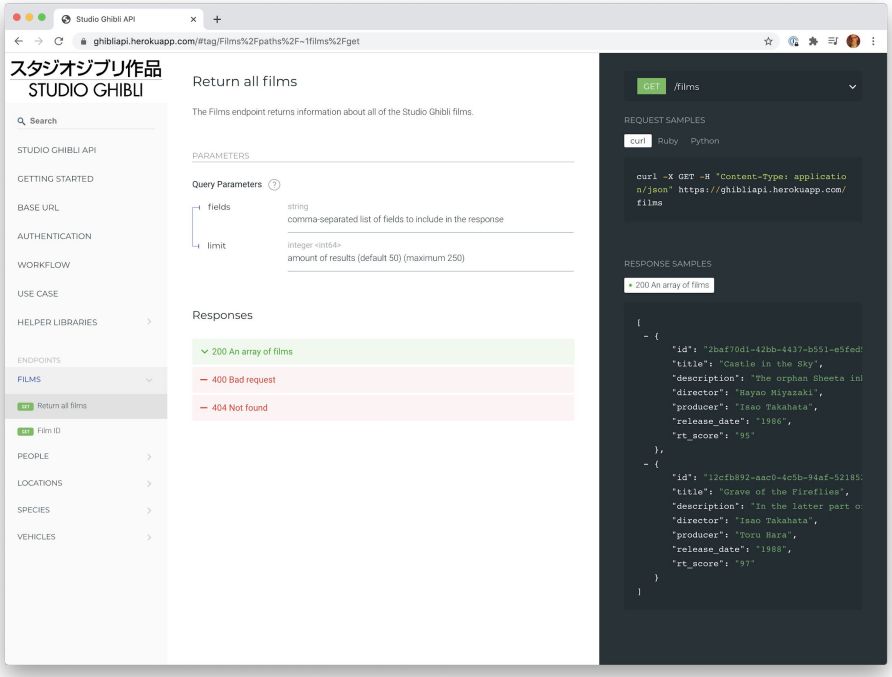
Studio Ghibli Project

For this project, we will use the Studio Ghibli API which provides information on movies created by this fantastic Japanese animation studio.



Studio Ghibli API Documentation

The place to start with any API is with the documentation, so you know what it can do and how to use it.



Accessing the Films

According to the API, you can go to this endpoint to access data on all the films:

https://ghibliapi.herokuapp.com/films

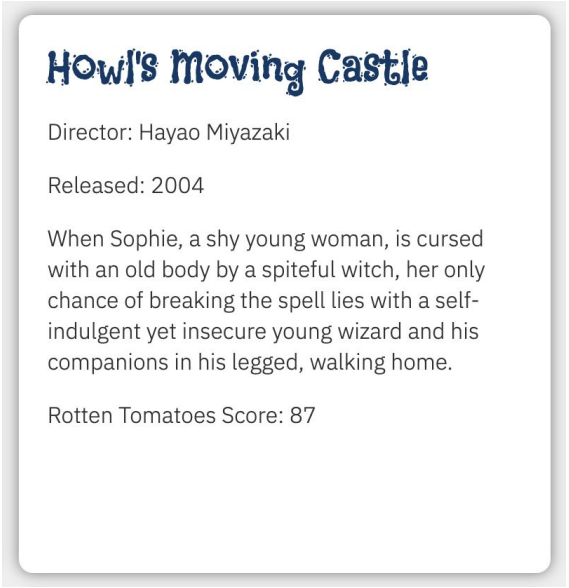
Try This To Start

```
const mainElement = document.querySelector('main');

async function getFilms() {
    const filmsPromise = await fetch('https://ghibliapi.herokuapp.com/films');
    const films = await filmsPromise.json();
    console.log(films);
}

getFilms();
```

You are not putting anything on the page yet with this, but start by getting the <main> element on the document and putting it into a variable. That is where this will go, shortly.

Creating the Card

The task is to create a card for each movie. If you look at the Studio Ghibli API for films, you will see the type of data you can get for each movie.



Creating The Card

Javascript will create the card, which is just an article element with the h2 set to be the heading. Create each element, create each text node and append it to the element, then append all the elements to the card, then append the card to the page.

```
function createCard(data) {
    const card = document.createElement('article');
    const movieTitle = document.createElement('h2');
    const movieTitleTxt = document.createTextNode(data.title);
    movieTitle.appendChild(movieTitleTxt);

    const director = document.createElement('p');
    const directorTxt = document.createTextNode(`Director: ${data.director}`);
    director.appendChild(directorTxt);

    //write similar code for year, description and score

    card.appendChild(movieTitle);
    card.appendChild(director);
    // append the year, description and score

    mainElement.appendChild(card);
}
```

The Whole Function

Here is a picture of the whole function you should have been able to finish from the previous slide.

```
function createCard(data) {
    const card = document.createElement('article');
    const movieTitle = document.createElement('h2');
    const movieTitleTxt = document.createTextNode(data.title);
    movieTitle.appendChild(movieTitleTxt);

    const director = document.createElement('p');
    const directorTxt = document.createTextNode(`Director: ${data.director}`);
    director.appendChild(directorTxt);

    const year = document.createElement('p');
    const yearTxt = document.createTextNode(`Released: ${data.release_date}`);
    year.appendChild(yearTxt);

    const description = document.createElement('p');
    const descriptionTxt = document.createTextNode(data.description);
    description.appendChild(descriptionTxt);

    const rating = document.createElement('p');
    const ratingTxt = document.createTextNode(`Rotten Tomatoes Score: ${data.rt_score}`);
    rating.appendChild(ratingTxt);

    card.appendChild(movieTitle);
    card.appendChild(director);
    card.appendChild(year);
    card.appendChild(description);
    card.appendChild(rating);

    mainElement.appendChild(card);

}
```

Updated getFilms() Function

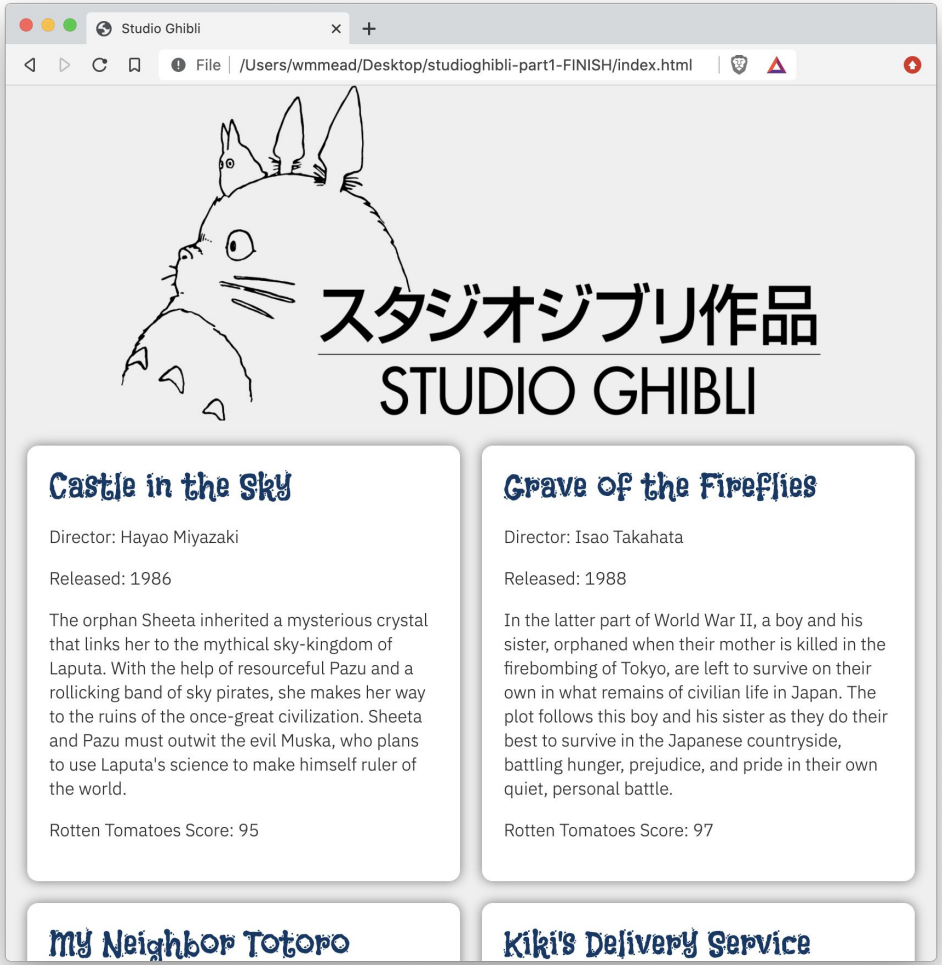
```
async function getFilms() {
    const filmsPromise = await fetch('https://ghibliapi.herokuapp.com/films');
    const films = await filmsPromise.json();

    films.forEach(eachFilm => {
        createCard(eachFilm);
    });
}
```

Did you get something like this?

The variable **`eachFilm`** represents each element in the array, which is, in fact a film. So a card is created and added to the <main> element for each film.

Results So Far

Since the CSS is already written, it should just display rather nicely.



Sorting the Films

```
async function getFilms() {
    const filmsPromise = await fetch('https://ghibliapi.herokuapp.com/films');
    const films = await filmsPromise.json();

    films.sort((a, b) => (a.title > b.title) ? 1 : -1);

    films.forEach(eachFilm => {
        createCard(eachFilm);
    });
}
```

Check out the additional line of code highlighted in the script.

Alphabetically Sorted

You can see the films are sorted alphabetically. However, you might want some choices in how they are sorted. For example you might want to sort by release year or by rating.

Add This Form

Add this nav element after the <header> on the index.html file.

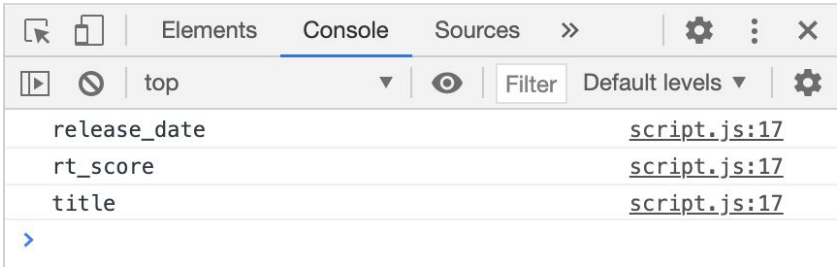
```
<nav>
    <form>
        <select id="sortorder">
            <option value="title">Title</option>
            <option value="release_date">Release Date</option>
            <option value="rt_score">Rotten Tomatoes Score</option>
        </select>
    </form>
</nav>
```

```
nav {
    padding: 20px;
    text-align: center;
}
```

Listening for Change

```
document.getElementById('sortorder').addEventListener('change', function () {
    console.log(document.getElementById('sortorder').value);
});
```

You need an event handler that captures when the user changes the value of the select list.



Sorting Function

```
function setSort(array) {
    const sortOrder = document.getElementById('sortorder').value;
    switch (sortOrder) {
        case 'title': array.sort((a, b) => (a.title > b.title) ? 1 : -1); break;
        case 'release_date': array.sort((a, b) => (a.release_date > b.release_date) ? 1 : -1); break;
        case 'rt_score': array.sort((a, b) => (a.rt_score >b.rt_score) ? 1 : -1); break;
    }
}
```

Next, you need a function that will take the value from the select list, and run a different sort based on that value.

This function is using a switch statement, but could just as easily use if/else if statements.

Part 1 - Running the Sort

Run the setSort() function inside the getFilms function. Pass in the films array.

When the page loads, the select list will be on the title option by default and the films will sort by title.

```
async function getFilms() {
    const filmsPromise = await
fetch('https://ghibliapi.herokuapp.com/films');
    const films = await filmsPromise.json();

    setSort(films);

    films.forEach(eachFilm => {
        createCard(eachFilm);
    });
}
```

Part 2 - Changing the Sort

```
document.getElementById('sortorder').addEventListener('change', function () {
    mainElement.innerHTML = '';
    getFilms();
});
```

Clear out all the HTML in the main element, then run getFilms(). Since the select list value has changed, a new sort will be passed in and the getFilms() function will get the films and apply the new sort to them.

Rotten Tomatoes Sort Fixed

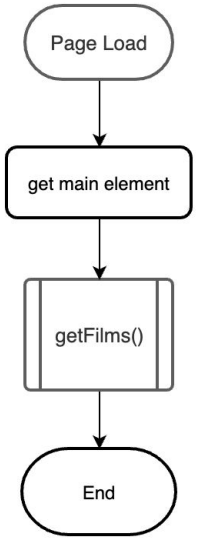
```
case 'rt_score': array.sort((a, b) => (parseInt(a.rt_score) > parseInt(b.rt_score)) ? -1 : 1); break;
```

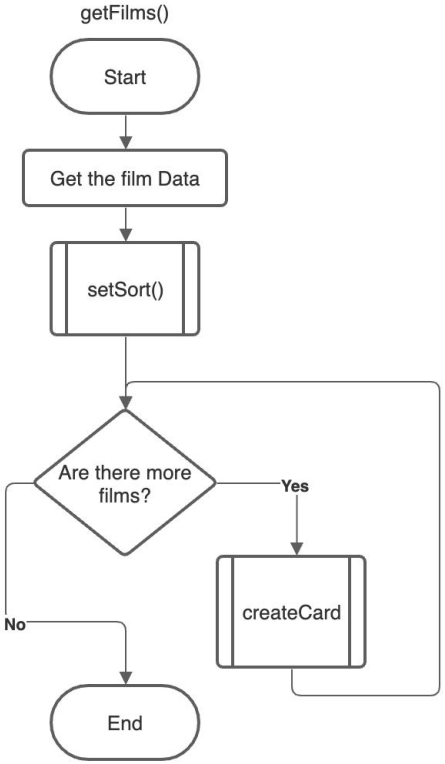
Did you solve it?

If you wrap the a.rt_score and b.rt_score with parseInt(),
it will convert those two strings to numbers.

The Current Flow

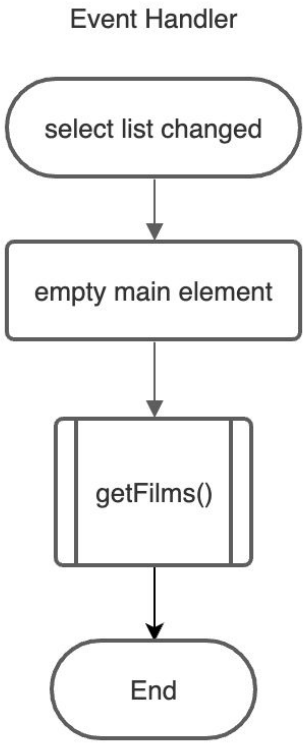
In the current flow, the page loads, we get the main element from the DOM and then run the getFilms() function, which gets the data, sets the sort order, and loops through the data, creating the cards and adding them to the page.





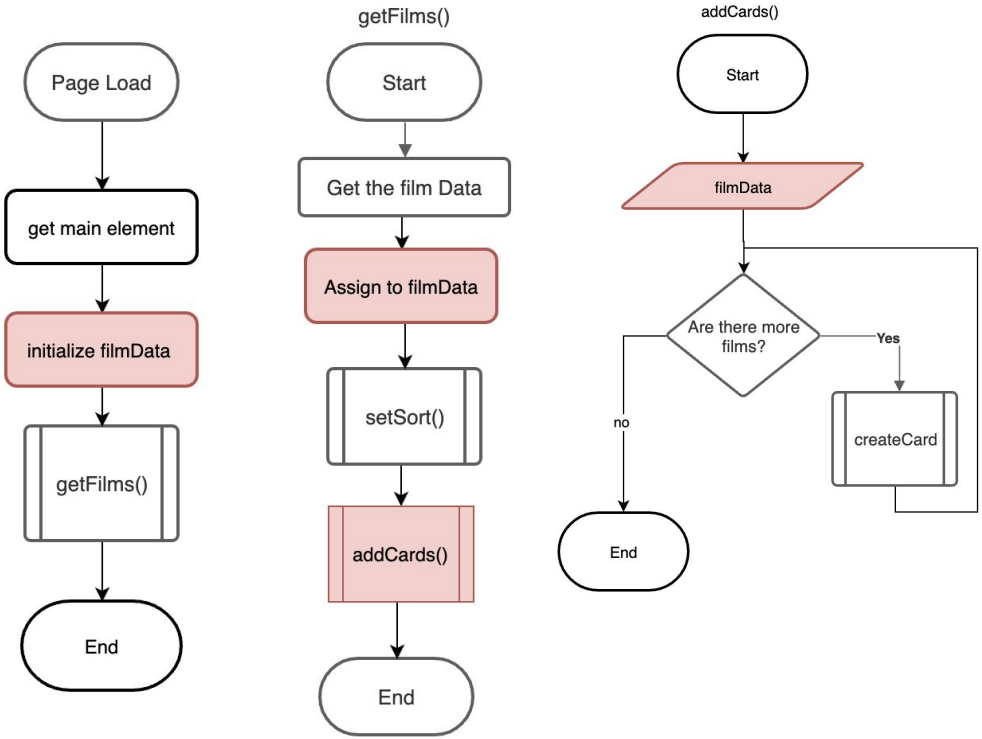
The Problem

The problem is that when the event handler detects a change in the select list, it runs getFilms() again (with a new sort), and that forces a reload of all the data from the API.



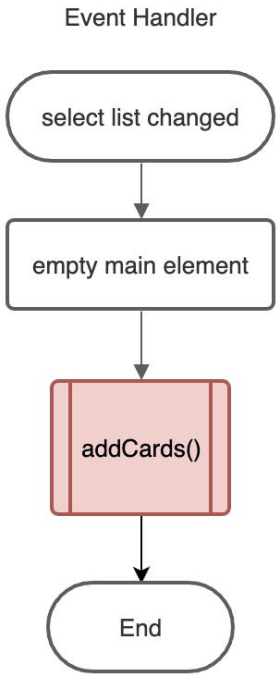
The Solution Strategy

- Initialize the variable, filmData
- Pull the loop out and create a function called addCards()



The Event Handler

Now the event handler can run addCards() instead of getFilms(). Since addCards() is pulling data from the filmData variable, there is no reason to go back to the API to get the data.



Add the filmData Variable

```
const mainElement = document.querySelector('main');

let filmData;

async function getFilms() {
```

Add the variable filmData, just after the variable for the mainElement. It has no value to start with, but you want it in the global scope of the script.

AddCards Function

Next add the addCards() function, which takes an array as an input. This is just a very simple function for looping through the array.

```
function addCards(array) {
    array.forEach(eachItem => {
        createCard(eachItem);
    });
}
```

Update the getFilms() Function

```
async function getFilms() {
    const filmsPromise = await fetch('https://ghibliapi.herokuapp.com/films');
    const films = await filmsPromise.json();

    setSort(films);
    addCards(films);
    filmData = films;
}
```

Update the getFilms() function with the addCards() function, instead of the forEach loop that adds all the cards. You pass in the films array. Then you assign that same films array to the filmData variable created in the global scope.

Update the Click Handler

```
document.getElementById('sortorder').addEventListener('change', function () {
    mainElement.innerHTML = '';
    setSort(filmData);
    addCards(filmData);
});
```

Update the click handler so that it is calling addCards() instead of getFilms().

Pass addCards() the variable with all the data in it and that will keep the script from going back to the API to get the data again.

One More Thing

It is possible, although unlikely, that a user could try to use the sort order select list *before* the data has loaded from the API. This would run the addCard() function, which passes in the filmData array, which would be empty at this time, therefore throwing an error.

In the HTML just add a disabled attribute to the select list, as shown here. Then, in the getFilms() function, at the very bottom, remove that attribute.

```
<form>
    <select id="sortorder" disabled>
        <option value="title">Title</option>
        <option value="release_date">Release Date</option>
        <option value="rt_score">Rotten Tomatos Score</option>
    </select>
</form>
```

```
document.getElementById('sortorder').removeAttribute('disabled');
```

Summary

In this project you have taken data available through a free API and consumed it with JavaScript, then used asynchronous functions to display the data on a web page.