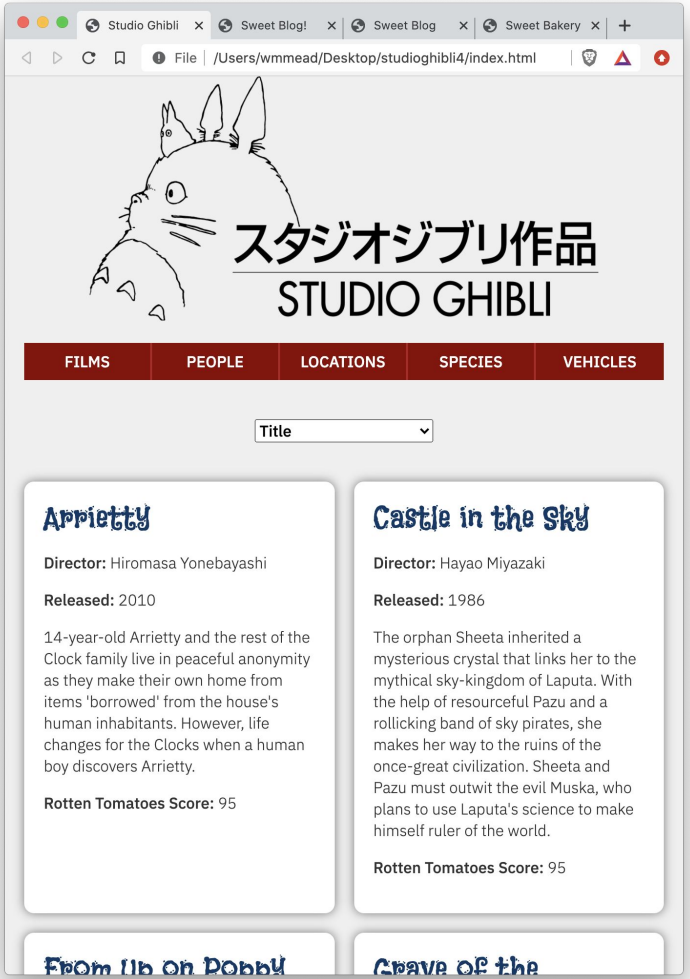
Studio Ghibli Part 2

More on 3rd Party APIs

Getting Started

There is more data to get from this API and doing more with it will be instructive.



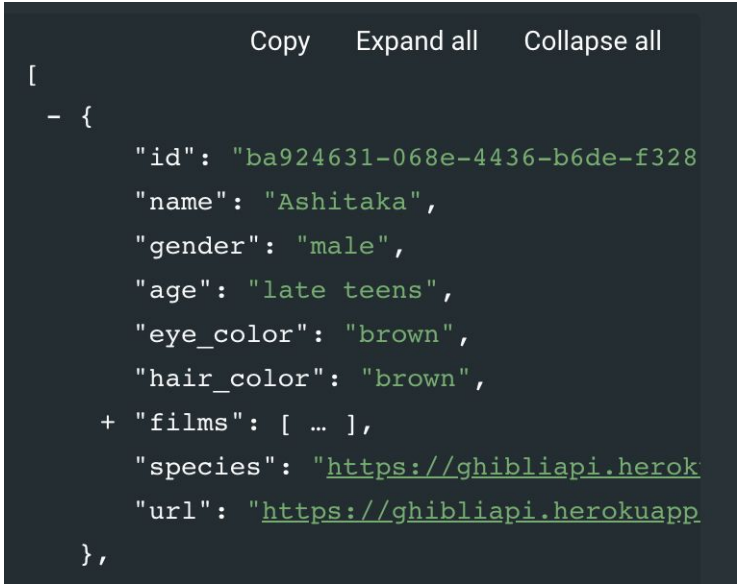
Added Data

This version has a horizontal navigation bar and some styling for that built into the HTML and CSS.

The Studio Ghibli API provides data for People, Locations, Species, and Vehicles.

Cards for Other Data

You can see in the Studio Ghibli API that when you click on people, you get a different set of data than you got for the films.



Updated createCard Function

```
function createCard(data) {
    const card = document.createElement('article');
    card.innerHTML = filmCardContents(data);
    mainElement.appendChild(card);
}

function filmCardContents(data) {
    let html = `<h2>${data.title}</h2>`;
    html += `<p><strong>Director:</strong> ${data.director}</p>`;
    html += `<p><strong>Released:</strong> ${data.release_date}</p>`;
    html += `<p>${data.description}</p>`;
    html += `<p><strong>Rotten Tomatoes Score:</strong> ${data.rt_score}</p>`;
    return html;
}
```

Here is the updated createCard() function, and a filmCardContents() function that will create the guts of the card.

Helper Function

If you look again at the data for individual people in the API, there is an array for films, but there's also a URL for the species of that person.

```
async function indivItem(url, item) {
    const itemPromise = await fetch(url);
    const data = await itemPromise.json();
    return data[item];
}
```

The Guts for the People Card

```
async function peopleCardContents(data) {
    const species = await indivItem(data.species, 'name');
    let html = `<h2>${data.name}</h2>`;
    html += `<p><strong>Details:</strong> gender ${data.gender}, age ${data.age}, eye color
${data.eye_color}, hair color ${data.hair_color}</p>`;
    html += `<p><strong>Species:</strong> ${species}</p>`;
    return html;
}
```

This function will create the guts for the people card, without the films (you will add those next).

Getting the Films

```
const thefilms = data.films;
let filmtitles = [];
for (eachFilm of thefilms) {
      const filmTitle = await indivItem(eachFilm, 'title');
      filmtitles.push(filmTitle);
}
```

To get the films, add this to the very top of the peopleCardContents() function.

```
html += `<p><strong>Films:</strong> ${filmtitles.join(', ')}</p>`;
```

The peopleCardContents Function

```
async function peopleCardContents(data) {
      const thefilms = data.films;
      let filmtitles = [];
      for (eachFilm of thefilms) {
            const filmTitle = await indivItem(eachFilm, 'title');
            filmtitles.push(filmTitle);
      }
    const species = await indivItem(data.species, 'name');
    let html = `<h2>${data.name}</h2>`;
    html += `<p><strong>Details:</strong> gender ${data.gender}, age ${data.age}, eye color
${data.eye_color}, hair color ${data.hair_color}</p>`;
      html += `<p><strong>Films:</strong> ${filmtitles.join(', ')}</p>`;
    html += `<p><strong>Species:</strong> ${species}</p>`;
    return html;
}
```

Testing the People Cards

```
async function createCard(data) {
    const card = document.createElement('article');
    //card.innerHTML = filmCardContents(data);
      card.innerHTML = await peopleCardContents(data);
    mainElement.appendChild(card);
}
```

To test the people cards, to see if they are working, make a few minor modifications to the script. First up is the createCard() function.

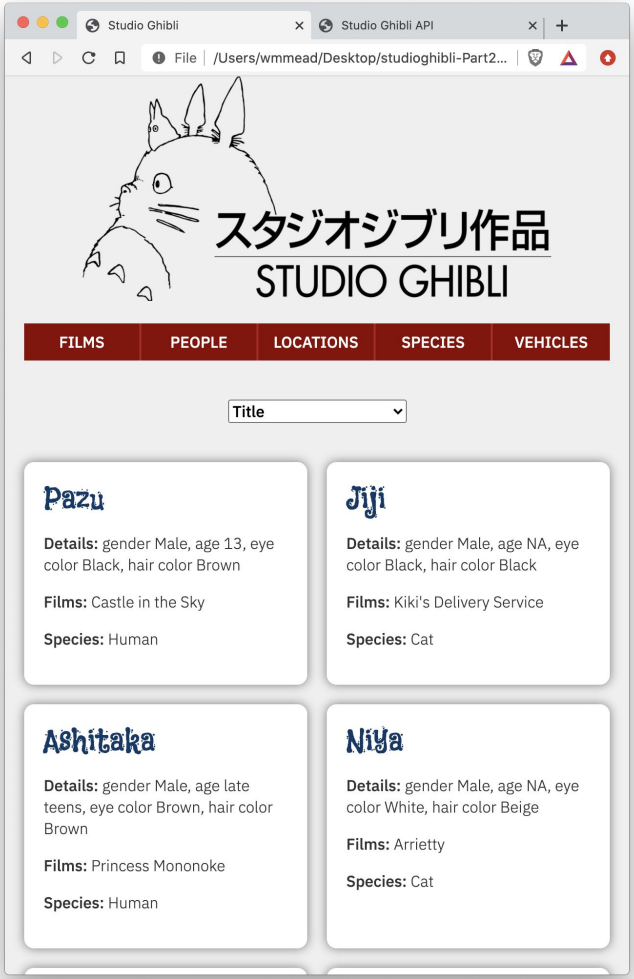
Temp Change to getFilms()

```
async function getFilms() {
    const filmsPromise = await fetch('https://ghibliapi.herokuapp.com/people');
    const films = await filmsPromise.json();
    //setSort(films);
    addCards(films);
    //filmData = films;
    document.getElementById('sortorder').removeAttribute('disabled');
}
```

The other thing you have to do to test this, is change the getFilms() function so that it gets the people instead.

Testing People!

Success! You should be getting a page of people!



Getting the Links

```
const navLinks = document.querySelectorAll('#mainnav ul li a');
```

Add this variable at the top of the script.

```
<nav id="mainnav">
    <ul>
        <li><a href="#films">Films</a></li>
        <li><a href="#people">People</a></li>
        <li><a href="#locations">Locations</a></li>
        <li><a href="#species">Species</a></li>
        <li><a href="#vehicles">Vehicles</a></li>
    </ul>
</nav>
```

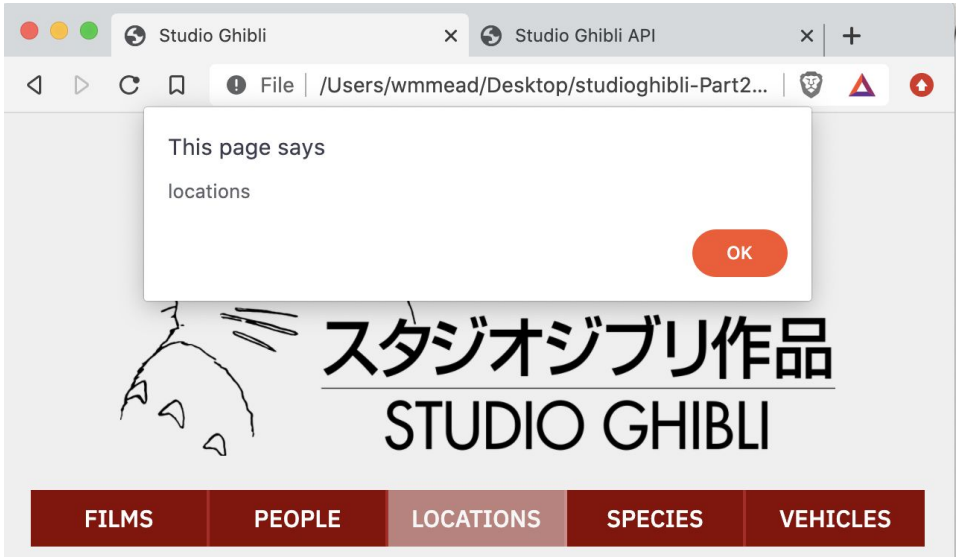
Add the Click Handlers

```
navLinks.forEach(function (eachLink) {
    eachLink.addEventListener('click', function (event) {
        event.preventDefault();
        const thisLink = event.target.getAttribute('href').substring(1);
        alert(thisLink);
    });
});
```

Add this click handler to the script file.

Getting Alerts

You should get an alert for each link.



Add Two Variables

```
const mainElement = document.querySelector('main');
const navLinks = document.querySelectorAll('#mainnav ul li a');

let filmData;
let dataSet = 'films';
let url = 'https://ghibliapi.herokuapp.com/films';
```

At the top of the script file, add two more variables. One for **`dataSet`**, which should be assigned “films” initially, and one for **`url`**, which will be set with the end point for the API for films, initially.

Challenge: Can you update the click handler to set these two variables according to which link was clicked?

Add Two Variables

```
navLinks.forEach(function (eachLink) {
    eachLink.addEventListener('click', function (event) {
        event.preventDefault();
        const thisLink = event.target.getAttribute('href').substring(1);
        url = 'https://ghibliapi.herokuapp.com/' + thisLink;
        dataSet = thisLink;
        getData(url);
    });
});
```

Did you get something like the two lines highlighted in yellow?

Change getFilms to getData

```
async function getData(url) {
    const dataPromise = await fetch(url);
    const data = await dataPromise.json();

    if (dataSet === 'films') {
        mainElement.innerHTML = '';
        setSort(data);
        addCards(data);
        filmData = data;
        document.getElementById('sortorder').removeAttribute('disabled');
    }
    else {
        mainElement.innerHTML = '';
        addCards(data);
    }
}

getData(url);
```

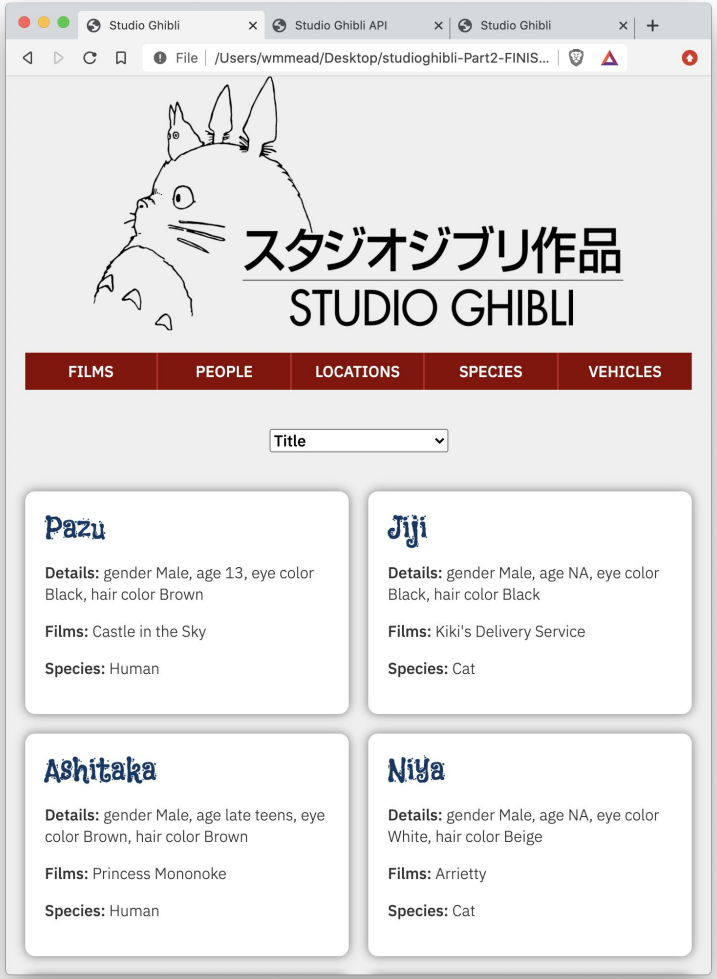
Update createCard() Function

```
async function createCard(data) {
    const card = document.createElement('article');
    switch (dataSet) {
        case 'films': card.innerHTML = filmCardContents(data); break;
        case 'people': card.innerHTML = await peopleCardContents(data); break;
    }
    mainElement.appendChild(card);
}
```

The last step to get this to work is to update the **`createCard()`** function so that it creates the right cards depending on which link is clicked.

Films and People Done

Clicking films and people in the navigation should be working now.



A Look at Locations

If you look at the locations data, it includes an array of residents in each location in the Studio Ghibli universe.


```
{
  "id": "6ba60a86-7c74-4ec4-a6f4-7112b5705a2f",
  "name": "Gondoa",
  "climate": "TODO",
  "terrain": "TODO",
  "surface_water": "40",
  "residents": [
    "TODO"
  ],
  "films": [
    "https://ghibliapi.herokuapp.com/films/2baf70d1-42bb-4437-b551-e5fed5a87abe"
  ],
  "url": [
    "https://ghibliapi.herokuapp.com/locations/6ba60a86-7c74-4ec4-a6f4-7112b5705a2f"
  ]
},
```

Locations Card Contents

```
async function locationCardContents(data) {
    const theResidents = data.residents;
    let residentNames = [];
    for (eachResident of theResidents) {
            const resName = await indivItem(eachResident, 'name');
            residentNames.push(resName);
    }

    const thefilms = data.films;
    let filmtitles = [];
    for (eachFilm of thefilms) {
        const filmTitle = await indivItem(eachFilm, 'title');
        filmtitles.push(filmTitle);
    }

    let html = `<h2>${data.name}</h2>`;
    html += `<p><strong>Details:</strong> climate ${data.climate}, terrain ${data.terrain}, surface water
${data.surface_water}%</p>`;
      html += `<p><strong>Residents:</strong> ${residentNames.join(', ')}</p>`;
    html += `<p><strong>Films:</strong> ${filmtitles.join(', ')}</p>`;
    return html;
}
```

Before Testing This

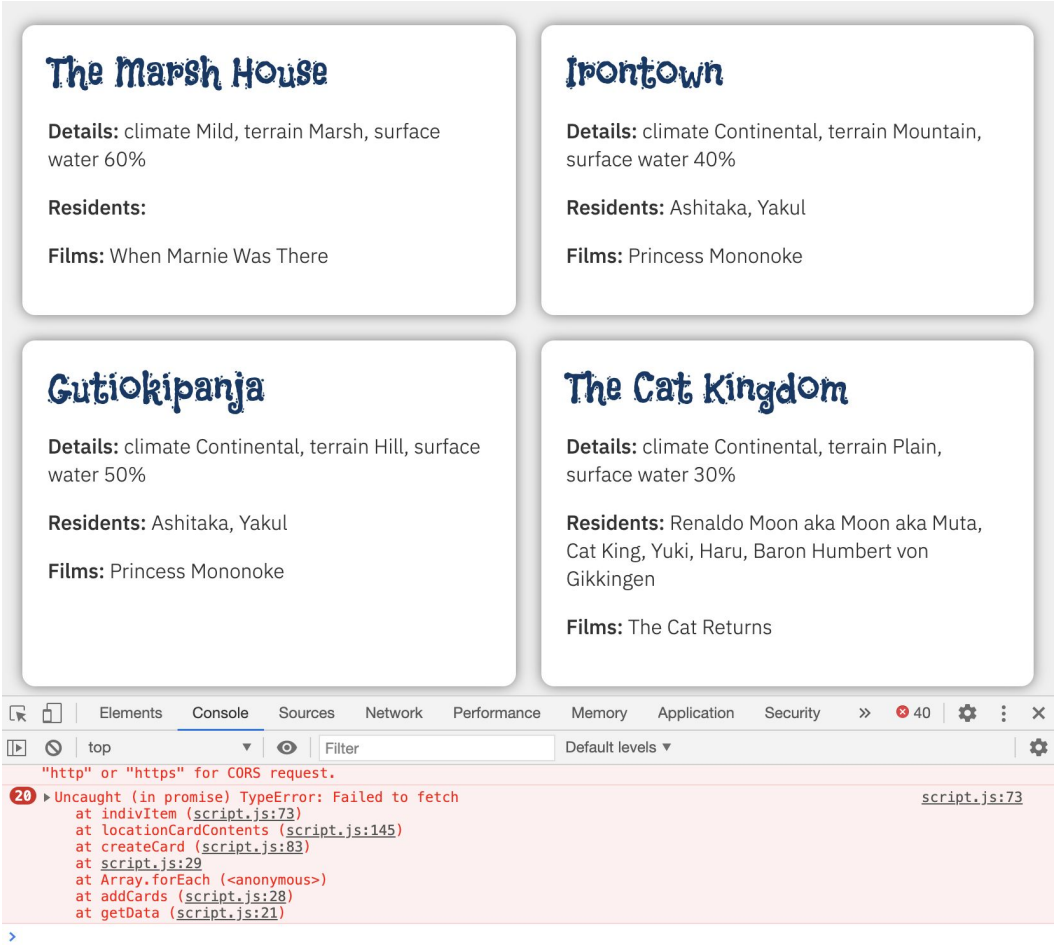
```
async function createCard(data) {
    const card = document.createElement('article');
    switch (dataSet) {
        case 'films': card.innerHTML = filmCardContents(data); break;
        case 'people': card.innerHTML = await peopleCardContents(data); break;
        case 'locations': card.innerHTML = await locationCardContents(data); break;
    }
    mainElement.appendChild(card);
}
```

Before testing this, be sure to add this line to the createCard() function. Once you save the file. Try it and go to the locations page. You will get a lot of errors.

Lots of Errors!

There are only four records without errors, and one of them contains an empty residents array. The rest contain “TODO” instead of a URL and result in an error.

It is a good idea to handle errors so that asynchronous functions don’t get hung up like this.



Updated indivItem() Function

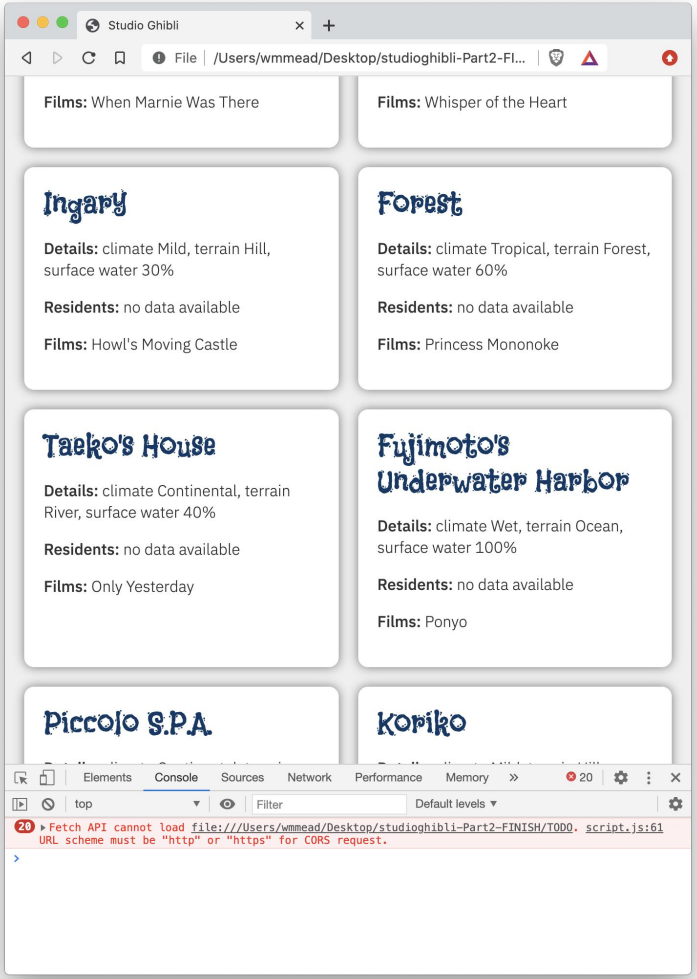
```
async function indivItem(url, item) {
      var theItem;
      try{
            const itemPromise = await fetch(url);
            const data = await itemPromise.json();
            theItem = data[item];
      } catch(err){
            theItem = "no data available";
      } finally{
            return theItem;
      }
}
```

One good way to deal with errors is to use the try, catch and finally structure. Here the function will try to get the data. If it fails, and it will, 20 times, the catch will fire and set the variable theItem to “no data available”.

Then finally runs and returns theItem, which either has the name of a resident (in this case) or it has “no data available”.

Still Errors

The errors still occur, but they do not hold up the script at all, and at least you get all your cards.



Update locationCardContents()

```
const regex = 'https?:\/\/';
const theResidents = data.residents;
let residentNames = [];
for (eachResident of theResidents) {
      if(eachResident.match(regex)){
            const resName = await indivItem(eachResident, 'name');
            residentNames.push(resName);
      }
      else {
            residentNames[0]='no data available';
      }
}
```

Replace the code at the top of this function dealing with the residents and add this instead.

CHALLENGE

Can you write the functions for the contents of the species and vehicle cards, and update the createCards() function to display them properly on your own?

Species

```
async function speciesCardContents(data) {
    const people = data.people;
    let peopleNames = [];
    for (eachPerson of people) {
        const personName = await indivItem(eachPerson, 'name');
        peopleNames.push(personName);
    }

    const thefilms = data.films;
    let filmtitles = [];
    for (eachFilm of thefilms) {
        const filmTitle = await indivItem(eachFilm, 'title');
        filmtitles.push(filmTitle);
    }

    let html = `<h2>${data.name}</h2>`;
    html += `<p><strong>Classification:</strong> ${data.classification}</p>`;
    html += `<p><strong>Eye Colors:</strong> ${data.eye_colors}</p>`;
    html += `<p><strong>Hair Colors:</strong> ${data.hair_colors}</p>`;
    html += `<p><strong>People:</strong> ${peopleNames.join(', ')}</p>`;
    html += `<p><strong>Films:</strong> ${filmtitles.join(', ')}</p>`;
    return html;
}
```

Vehicles

```
async function vehicleCardContents(data) {
    let html = `<h2>${data.name}</h2>`;
    html += `<p><strong>Description:</strong> ${data.description}</p>`;
    html += `<p><strong>Vehicle Class:</strong> ${data.vehicle_class}</p>`;
    html += `<p><strong>Length:</strong> ${data.length} feet</p>`;
    html += `<p><strong>Pilot:</strong> ${await indivItem(data.pilot, 'name')}</p>`;
    html += `<p><strong>Film:</strong> ${await indivItem(data.films, 'title')}</p>`;
    return html;
}
```

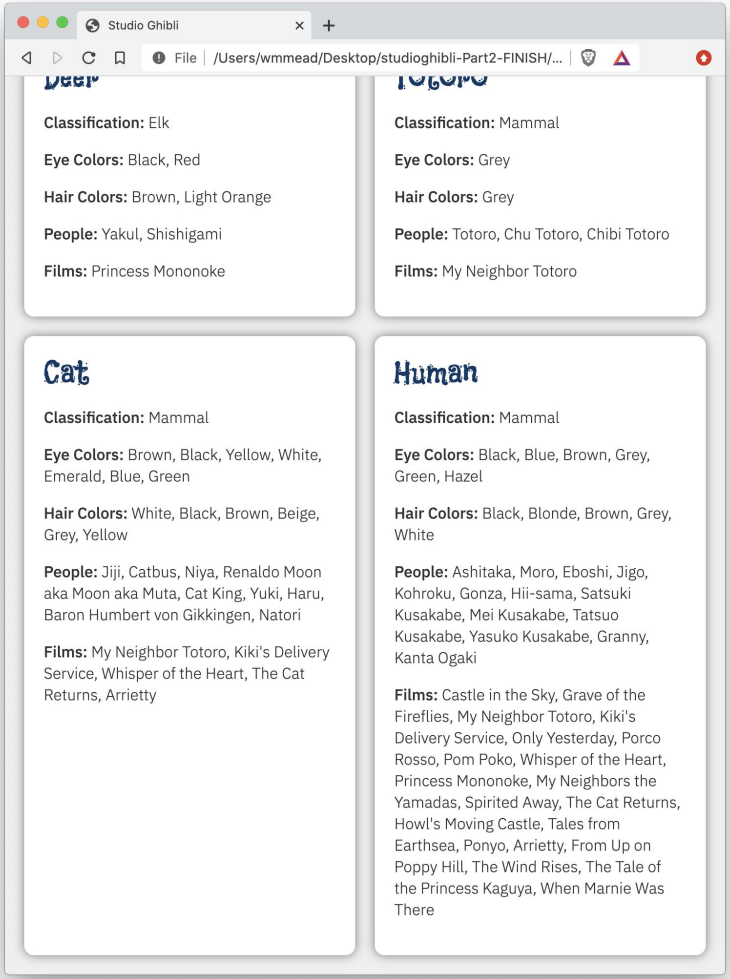
Updated createCards() Function

```
async function createCard(data) {
    const card = document.createElement('article');
    switch (dataSet) {
        case 'films': card.innerHTML = filmCardContents(data); break;
        case 'people': card.innerHTML = await peopleCardContents(data); break;
        case 'locations': card.innerHTML = await locationCardContents(data); break;
        case 'species': card.innerHTML = await speciesCardContents(data); break;
        case 'vehicles': card.innerHTML = await vehicleCardContents(data); break;
    }

    mainElement.appendChild(card);
}
```

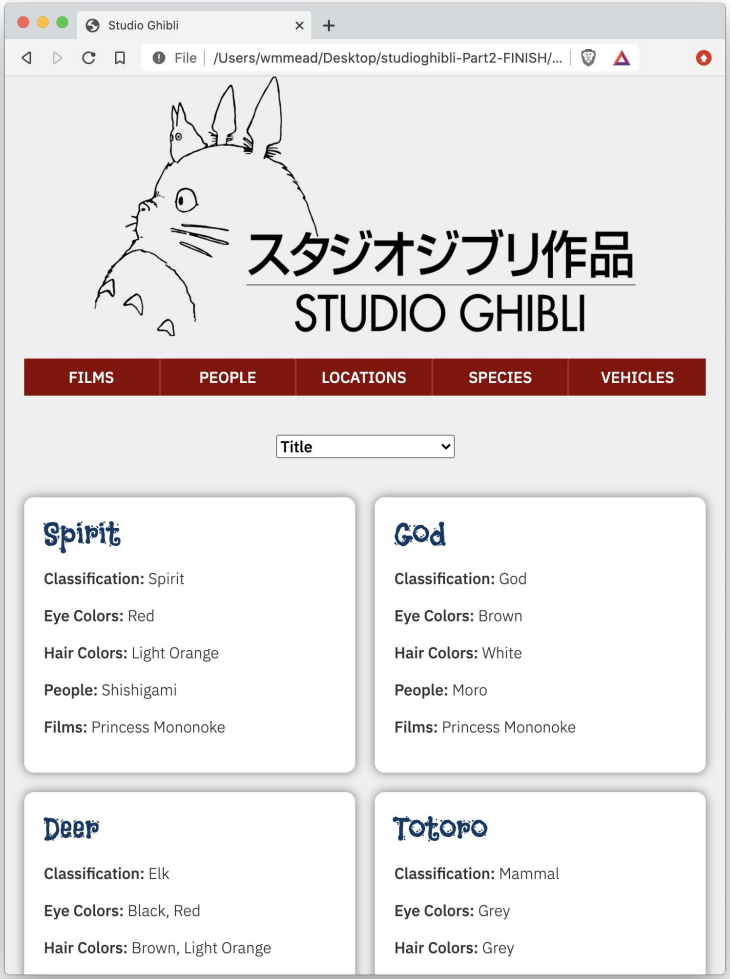
It's Working!

The pages are all working. However, notice when you click on the species page, it takes a while for the cat and the human to load.



Fixing the Sort

The sort functionality will cause an error on any other page than the films page.



Hiding the Sort Selector

```
async function getData(url) {
    const dataPromise = await fetch(url);
    const data = await dataPromise.json();

    if (dataSet === 'films') {
        mainElement.innerHTML = '';
        setSort(data);
        addCards(data);
        filmData = data;
        document.querySelector('nav form').style.visibility = "visible";
        document.getElementById('sortorder').removeAttribute('disabled');
    }
    else {
        mainElement.innerHTML = '';
        document.querySelector('nav form').style.visibility = "hidden";
        addCards(data);
    }
}
```

Here is my solution. I updated the getData() function to set that element hidden if you go to another page, and visible if you go to the films page.

Summary

That is a big project!

There are a lot of ways you could extend this project and make it even more interesting.

What else could you do with it?