CTD Intro Week 15

The Fetch API

Promises

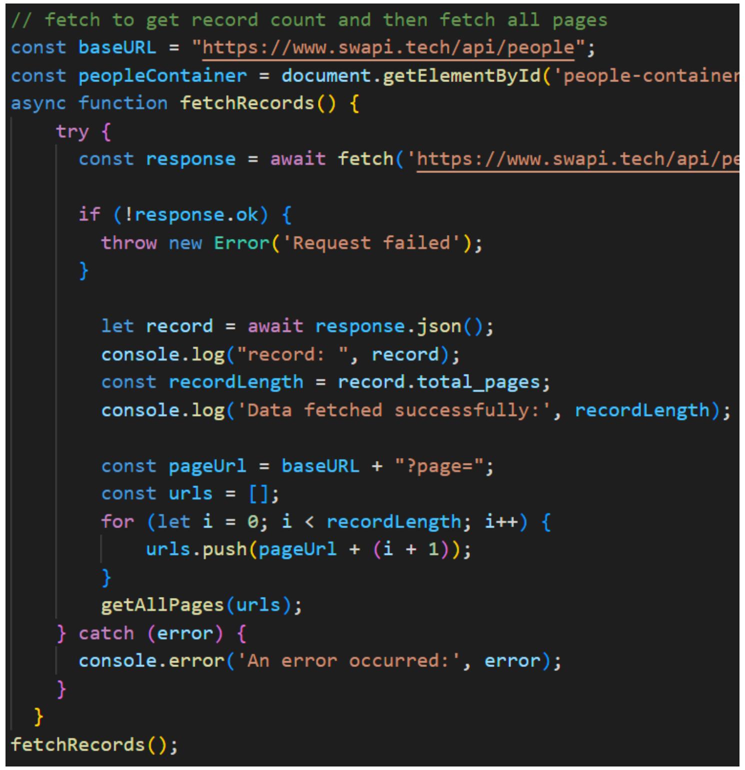
- Convenient way to handle asynchronous execution
  - States: pending, fulfilled, rejected
- ’then’ method specifies callback function(s)
  - resolve, reject
  - Asynchronous, executed on state change
- Chaining
  - myPromise.then().then()… .catch()
  - Next ‘then’ executes after previous ‘then’ resolve/reject
  - ‘catch’ can be chained to handle errors
- Await can be used to make synchronous
  - Not usually done, worse performance

Fetch API

- ‘fetch(url[, options])’ creates a promise
  - resolve, reject callbacks as with all promises
- No options defaults to ‘get’
- Options is a configuration object
- method: The HTTP method default is GET.
- headers: HTTP headers to send
- body: The body of the request. This can be a string, a binary data, or a JSON object.
- credentials: A Boolean value that specifies whether or not you want to send the request with credentials. The default value is false.
- mode: A string that specifies the mode of the request. The default value is "cors".

Async and Await

- Async functions return a promise
- Await stops execution in the current thread until an asynchronous operation completes
    - Limited to use inside async functions or top-level modules
- They can be used together to simplify asynchronous operations
- Simplest to do your work inside the async function
- Prefer try/catch inside async functions
    - Simplifies code, fewer nested functions

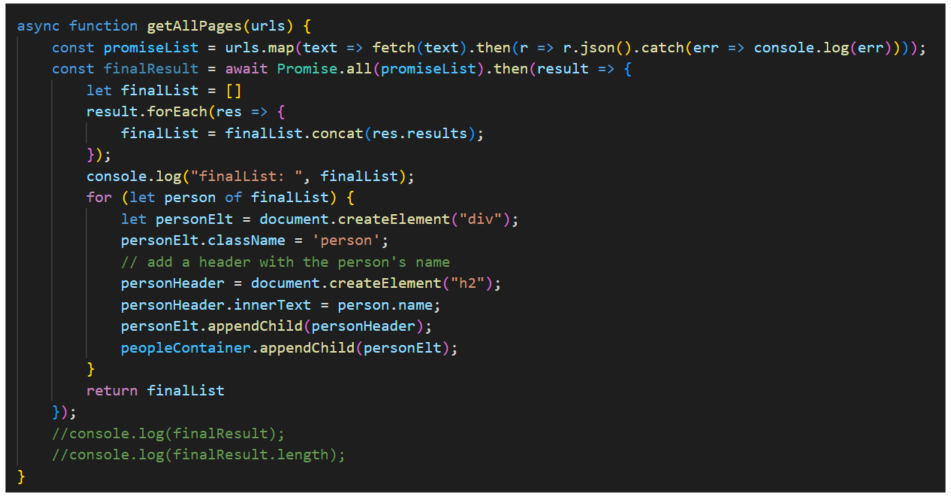


```
// fetch to get record count and then fetch all pages
const baseURL = "https://www.swapi.tech/api/people";
const peopleContainer = document.getElementById('people-container
async function fetchRecords() {
    try {
      const response = await fetch('https://www.swapi.tech/api/pe

      if (!response.ok) {
        throw new Error('Request failed');
      }

        let record = await response.json();
        console.log("record: ", record);
        const recordLength = record.total_pages;
        console.log('Data fetched successfully:', recordLength);

        const pageUrl = baseURL + "?page=";
        const urls = [];
        for (let i = 0; i < recordLength; i++) {
            urls.push(pageUrl + (i + 1));
        }
        getAllPages(urls);
    } catch (error) {
      console.error('An error occurred:', error);
    }
  }
fetchRecords();
```



```
async function getAllPages(urls) {
    const promiseList = urls.map(text => fetch(text).then(r => r.json().catch(err => console.log(err))));
    const finalResult = await Promise.all(promiseList).then(result => {
        let finalList = []
        result.forEach(res => {
            finalList = finalList.concat(res.results);
        });
        console.log("finalList: ", finalList);
        for (let person of finalList) {
            let personElt = document.createElement("div");
            personElt.className = 'person';
            // add a header with the person's name
            personHeader = document.createElement("h2");
            personHeader.innerText = person.name;
            personElt.appendChild(personHeader);
            peopleContainer.appendChild(personElt);
        }
        return finalList
    });
    //console.log(finalResult);
    //console.log(finalResult.length);
}
```

Open API Project

- Separate project in Github
  - Clone to a new location outside your current local repo.
- Suggested APIs in the lesson, but use any which is free and open
- At least two pages and endpoints with navigation buttons.
  - Requirements under final project
- For this week, just boilerplate and test the fetch.
- Put the repo name in your assignment submissions
  - Will also show up under projects!

Questions and Demo