Modified slightly from https://homework.adhoc.team/fetch/

Fetch

At the NYC Digital Service, our front-end applications often exchange JSON data with various RESTful APIs. We use modern JavaScript to interact with these APIs and to transform their responses into a format needed by the client application. In this homework, we provide a sample API with a single endpoint and ask you to write some JavaScript to request data from the API and transform the response.

Records API

You'll be provided with a `/records` API endpoint that returns a JSON array of items in the following format:

```
[
  {
    "id": 1,
    "color": "brown",
    "disposition": "closed"
  },
  {
    "id": 2,
    "color": "green",
    "disposition": "open"
  }
]
```

Each item returned from the `/records` endpoint will have the following:

- id: A unique integer
- color: One of `"red"`, `"brown"`, `"blue"`, `"yellow"`, or `"green"`
- disposition: Either `"open"` or `"closed"`

The `/records` endpoint accepts the following options, sent as query string parameters on the request URL:

- limit: The number of items to be returned
- offset: The index of the first item to be returned
- color[]: Which color to be included in the result set. May be included multiple times, once for each color. If omitted, all colors will be returned.

An example request URL might look like:

```
/records?limit=2&offset=0&color[]=brown&color[]=green
```

Task

In `api/managed-records.js`, write a function named `retrieve` that requests data from the `/records` endpoint, transforms the result into the format outlined below, and returns a promise that resolves with the transformed object. In addition to `retrieve`, you may add as many helper functions as you wish.

1. Get data from the `/records` endpoint using the Fetch API. Process pages of 10 items at a time. Note that the `/records` endpoint may return more than 10 items per request.
2. The `retrieve` function accepts an `options` object and should support the following keys:

- page - Specifies which page to retrieve from the `/records` endpoint. If omitted, fetch page 1.
- colors - An array of colors to retrieve from the `/records` endpoint. If omitted, fetch all colors.
  As an example, to fetch the 2nd page of red and brown items from the API, `retrieve` might be called like this:

```
retrieve({ page: 2, colors: ["red", "brown"] });
```

3. You can assume standard HTTP status codes on the response. If a request is unsuccessful, output a simple error message via `console.log()` and recover.
4. Upon a successful API response, transform the fetched payload into an object containing the following keys:

- ids: An array containing the ids of all items returned from the request.
- open: An array containing all of the items returned from the request that have a `disposition` value of `"open"`. Add a fourth key to each item called `isPrimary` indicating whether or not the item contains a primary color (red, blue, or yellow).

- closedPrimaryCount: The total number of items returned from the request that have a `disposition` value of `"closed"` and contain a primary color.
- previousPage: The page number for the previous page of results, or `null` if this is the first page.
- nextPage: The page number for the next page of results, or `null` if this is the last page.

5. Return a promise from `retrieve` that resolves with the transformed data.

Additional requirements

- Use the provided URI library to construct the request URL. Refer to https://medialize.github.io/URI.js/ for documentation.
- Write your solution using ES2015 or later. Any methods and syntax supported by the Babel polyfill are considered fair game. https://babeljs.io/docs/usage/polyfill/
- You must use the provided Fetch API to interact with the `records` endpoint. Refer to https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API for documentation.
- Don't add any additional libraries or edit any files other than `api/managed-records.js`

- We've provided a suite of unit tests. Your solution should pass all tests.
- Please delete the `node_modules` directory before submitting your completed homework.

Setup

Requirements: NodeJS >= 10, yarn

`yarn install` to install.

`yarn test` to run the provided unit tests.

Included files

The files for this assessment are included in FrontEnd.zip