Obtaining data for our application

```javascript
import React, { useState, useEffect } from 'react';
import { List } from 'immutable';
import filmData from './films.json';
import FilmTableContainer from './components/FilmTableContainer';
import SearchBar from './components/SearchBar';

function FilmExplorer() {
  const [searchTerm, setSearchTerm] = useState('');
  const [sortType, setSortType] = useState('title');
  const [films, setFilms] = useState(List());

  useEffect(() => {
    setFilms(List(filmData));
  }, []);

  // load the film data
  useEffect(() => {
    setFilms(List(filmData));
  }, []);
}
```

We will use `window.fetch` to obtain data asynchronously.

HTTP (and URLs)

HTTP request includes: a method, URI, protocol version and headers

```
GET http://srch.com:80/main/search?q=cloud&lang=en#top
```

HTTP response includes: Protocol version and status code, headers, and body

```
2** OK
3** Resource moved
4** Forbidden
5** Error
```
HTTP methods (verbs)

<table>
<thead>
<tr>
<th>Method</th>
<th>Typical Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request a resource. Form fields can be sent as the query parameters.</td>
</tr>
<tr>
<td>HEAD</td>
<td>Similar to GET, but for just the response headers</td>
</tr>
<tr>
<td>POST</td>
<td>Send data to the server. Unlike GET, the data is transmitted in the request body. Action is up to server, but often creates a subordinate resource. The response may be a new resource, or just a status code.</td>
</tr>
<tr>
<td>PUT</td>
<td>Similar to POST, expect that PUT is intended to create or modify the resource at the specified URL, while POST creates or updates a subordinate resource.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete the specified resource</td>
</tr>
<tr>
<td>PATCH</td>
<td>Partial replacement of a resource, as opposed to PUT which specifies complete replacement.</td>
</tr>
</tbody>
</table>

REST (Representational State Transfer)

- An architectural style (rather than a standard)
  1. API expressed as actions on specific resources
  2. Use HTTP verbs as actions (in line with meaning in spec.)
  3. Responses can include hyperlinks to discover additional RESTful resources (HATEOAS)
- A RESTful API uses this approach (more formally, observes 6 constraints in R. Fielding’s 2000 thesis)
- “a post hoc [after the fact] description of the features that made the Web successful”*

Film Explorer API

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /api/films</td>
<td>List (read) all films</td>
</tr>
<tr>
<td>GET /api/films/:id</td>
<td>Read data from films with id == :id</td>
</tr>
<tr>
<td>PUT /api/films/:id</td>
<td>Update film with id == :id from request data</td>
</tr>
</tbody>
</table>

$ curl http://basin.cs.middlebury.edu:5042/api/films/340382
{"id":340382, "overview":"The film follows the story started in the first Attack on Titan live-action film.", "release_date":"2015-09-19", "poster_path":"acIG1tjNHbLP2GnlaW33SXC95S1.jpg", "title":"Attack on Titan: End of the World", "vote_average":4.2, "rating":5, "genres":[{"id":18,"filmId":340382},{"id":14,"filmId":340382},{"id":28,"filmId":340382},{"id":878,"filmId":340382}], "genre_ids":[18,14,28,878]}

CRUD(L) on a RESTful resource

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /api/films</td>
<td>Create new film from request data</td>
</tr>
<tr>
<td>GET /api/films/:id</td>
<td>Read data of film with id == :id</td>
</tr>
<tr>
<td>PUT /api/films/:id</td>
<td>Update film with id == :id from request data</td>
</tr>
<tr>
<td>DELETE /api/films/:id</td>
<td>Delete film with id == :id</td>
</tr>
<tr>
<td>GET /api/films</td>
<td>List (read) all films</td>
</tr>
</tbody>
</table>

A “route” maps <HTTP method, URL> to a controller action

*Rosenberg and Mateos, “The Cloud at Your Service” 2010*
Other features of REST APIs

- Resources can be nested
  
  GET /courses/3971/assignments/43746
  Assignment 0 in CS101 S19 on Canvas

- Think broadly about what is a resource
  
  GET /films/search?q=Jurassic
  Resource is a “search result list” matching query

  GET /films/34082/edit
  Resource is a form for updating film 34082 (form submit launches POST/PUT request)

Recall that the browser is asynchronous

A common action is to set state

```javascript
someAsyncOperation(someParams, (result, error) => {
  // Do something with the result or error
  newAsyncOperation(newParams, (result, error) => {
    // Do something more...
    });
  });
});
```

The “next” promise will be fulfilled with the result of the then handler

```
Promise.then((result) => {
  // Do something with the result
  return newAsyncOperation(newParams);
}).then((result) => {
  // Do something more
}).catch((error) => {  // Handle error});
```
Obtaining film data in Film Explorer

```javascript
const prom1 = fetch(`/api/films/`);
const prom2 = prom1.then((response) => {
  return response.json();
});
prom2.then((data) => {
  setFilms(data);
});
// Do something after
```

**Diagram:**
- `prom1`: Fetch data in background
- `prom2`: Parse JSON in background
- Time
- `response`: Parse and return response as JSON
- `data`: Set films