Obtaining data for our application

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

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

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

  return (...
```
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

POST  http://localhost:3000/movies/3

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)
  - API expressed as actions on specific resources
  - Use HTTP verbs as actions (in line with meaning in spec.)
  - 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 https://filmexplorer-server.profandrews.repl.co/api/films/1891  
{"adult":false,"backdrop_path":"/dMZxEdrWIzUmUoOz2zvmFuutbj7.jpg","genre_ids": [12,28,878],"id":1891,"original_language":"en","original_title":"The Empire Strikes Back","overview":"The epic saga continues as Luke Skywalker, in hopes of defeating the evil Galactic Empire, learns the ways of the Jedi from aging master Yoda. But Darth Vader is more determined than ever to capture Luke. Meanwhile, rebel leader Princess Leia, cocky Han Solo, Chewbacca, and droids C-3PO and R2-D2 are thrown into various stages of capture, betrayal and despair.","popularity":28.115,"poster_path":"/7Bu9itoGrLExs2Y2SeMNQ1vk2no.jpg","release_date":"1980-05-20","title": "The Empire Strikes Back","video":false,"vote_average":8.4,"vote_count":12692}  

*Rosenberg and Mateos, “The Cloud at Your Service” 2010*
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

Other features of REST APIs

- Resources can be nested
  - GET /courses/248180/assignments/1044477
    - Assignment 1 on Gradescope

- Think broadly about what is a resource
  - GET /movies/search?q=Jurassic
    - Resource is a “search result list” matching query
  - GET /movies/34082/edit
    - Resource is a form for updating movie 34082 (form submit launches POST/PUT request)

CS 312 Software Development

Fetching data: fetch and Promises

Recall that the browser is asynchronous

Web APIs

- DOM
- AJAX
- Timeout

Event Loop

Callback Queue
fetch returns a Promise

A common action is to set state

Obtaining film data in Film Explorer

fetch('/api/films/')
  .then((response) => {
    if (!response.ok) {
      throw new Error(response.statusText);
    }
    return response.json();
  })
  .then((data) => {
    setFilms(data);
  })
  .catch(err => console.log(err));

const response = await fetch('/api/films/');
if (!response.ok) {
  throw new Error(response.statusText);
}
const data = await response.json();
setFilms(data);
Obtaining film data in Film Explorer
now using await...

```javascript
const getData = async () => {
    const response = await fetch('/api/films/');
    if (!response.ok) {
        throw new Error(response.statusText);
    }
    const data = await response.json();
    setFilms(data);
}

getData();
// do something else
```

Effect hooks

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

// load the film data
useEffect(() => {
    const getData = async () => {
        const response = await fetch('/api/films/');
        if (!response.ok) {
            throw new Error(response.statusText);
        }
        const data = await response.json();
        setFilms(data);
    }
    getData();
}, []);
```