Obtaining data for our application

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

HTTP (and URLs)

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

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 the 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”

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

Film Explorer API

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/films</td>
</tr>
<tr>
<td>GET</td>
<td>/api/films/:id</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/films/:id</td>
</tr>
</tbody>
</table>

CRUD(L) on a RESTful resource

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/api/films</td>
</tr>
<tr>
<td>GET</td>
<td>/api/films/:id</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/films/:id</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/films/:id</td>
</tr>
<tr>
<td>GET</td>
<td>/api/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/3971/assignments/43746
    - Assignment 0 in CS101 S19 on Canvas
- 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)

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

```javascript
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));
```

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

Obtaining film data in Film Explorer

Response object with status, headers, and response body

Parse and return response as JSON
const prom1 = fetch('/api/films/');
const prom2 = prom1.then((response) => {
    return response.json();
});
prom2.then((data) => {
    setFilms(data);
})
// Do something after

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

// Do something after

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();

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

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

// 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();
}, []);