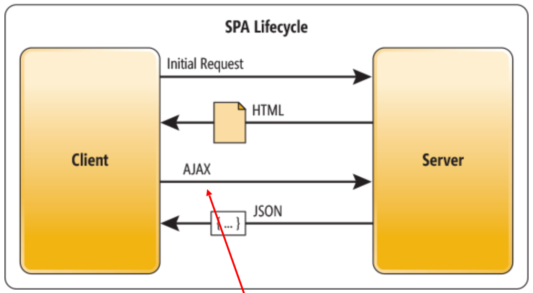
CS 312 Software Development

Fetching data: REST

Obtaining data for our application

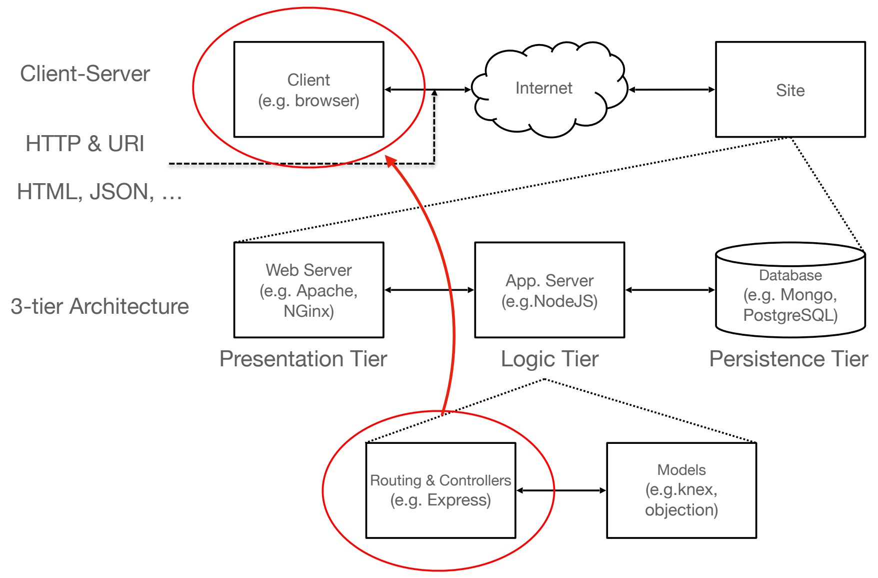
```
import React, { useState, useEffect } from 'react';

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);
  }, []);
```

Obtaining data for our application



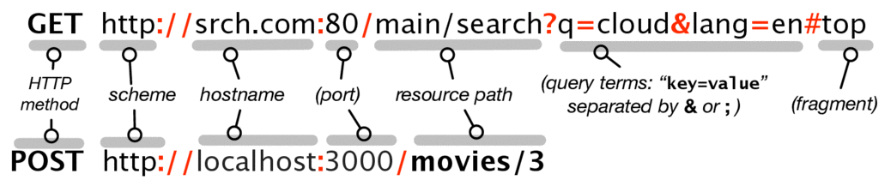
We will use `window.fetch` to obtain data asynchronously

Wasson, Microsoft



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 method
scheme
hostname
(port)
resource path
(query terms: "key=value" separated by & or ;)
(fragment)
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)

Method	Typical Use
GET	Request a resource. Form fields can be sent as the query parameters.
HEAD	Similar to GET, but for just the response headers
POST	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.
PUT	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.
DELETE	Delete the specified resource
PATCH	Partial replacement of a resource, as opposed to PUT which specifies complete replacement.

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

Route		Controller Action
GET	/api/films	List (read) all films
GET	/api/films/:id	Read data from films with id == :id
PUT	/api/films/:id	Update film with id == :id from request data

```
$ 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":"/
7BuH8itoSrLExs2YZSsM01Qk2no.jpg","release_date":"1980-05-20","title":
"The Empire Strikes
Back","video":false,"vote_average":8.4,"vote_count":12692}
```

CRUD(L) on a RESTful resource

Resource and "action"

Route	Controller Action	
POST /api/films	Create new film from request data	C
GET /api/films/:id	Read data of film with id == :id	R
PUT /api/films/:id	Update film with id == :id from request data	U
DELETE /api/films/:id	Delete film with id == :id	D
GET /api/films	List (read) all films	L

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

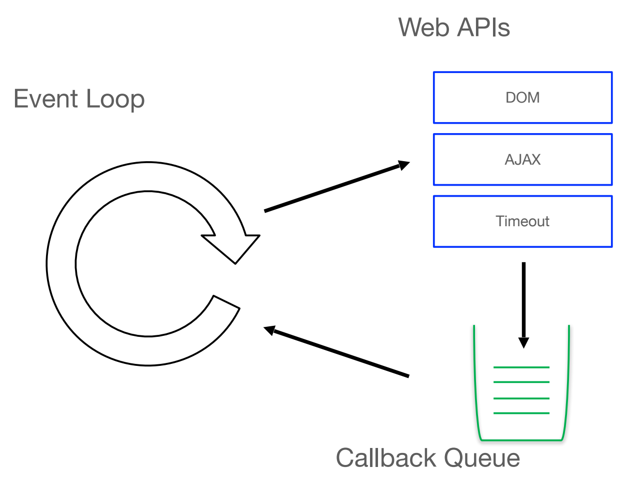
Other features of REST APIs

- 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

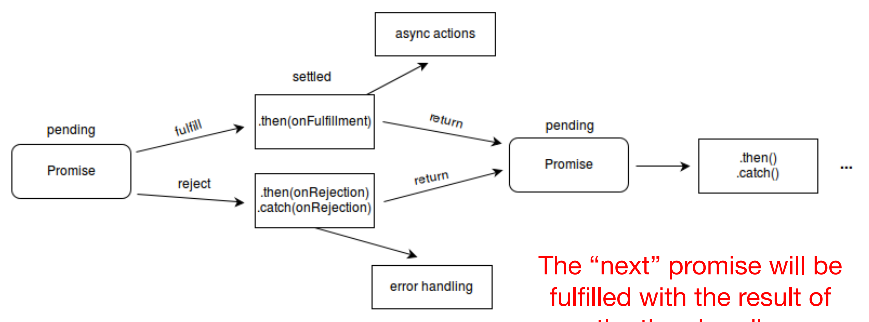
Fetching data: fetch and Promises

Recall that the browser is asynchronous



`fetch` returns a Promise

A common action is to set state



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

MDN

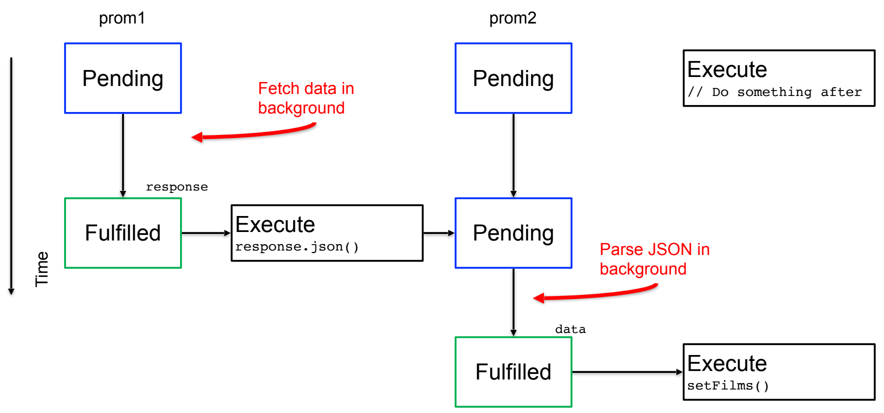
Obtaining film data in Film Explorer

```
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));
```

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
```



Obtaining film data in Film Explorer

now using await...

```
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…

```
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

```
// 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();
 }, []);
```