COMPSCI 326 - Web Programming
REST, & Fetch

join on the Slack #q-and-a channel as well as Zoom
Remember, you can ask questions of your teammates on your group Slack!
please **turn on your webcam** if you can
mute at all times *when you aren't asking a question*
(https://docs.google.com/document/d/1_DawWP1QCnsotgMq7wH13rzi8UnVUPaVHkByouW_p8A/edit?usp=sharing)

Today: REST & Fetch

resources:

- Concurrency model and the event loop - JavaScript | MDN
- Window setTimeout() Method
- Using Promises - JavaScript | MDN
- async function expression - JavaScript | MDN
- Using Fetch - Web APIs | MDN
- Video: What the heck is the event loop anyway? | Philip Roberts | JSConf EU

Last time: Promises + Async/Await
Today: actually fetching things, REST APIs

REST APIs

Quite common now for many sites to provide an API (Application Programmer Interface) to access info:

- Github: https://docs.github.com/en/free-pro-team@latest/rest/overview/resources-in-the-rest-api
- Twitter: https://developer.twitter.com/en/docs/twitter-api/getting-started/guide
- Rotten Tomatoes: https://developer.fandango.com/rotten_tomatoes
- Open Movie API: http://www.omdbapi.com/
- So many more!

These APIs are typically "RESTful". REST = "Representational state transfer". This means:

- There is some base URL, like https://api.github.com/
- You access it over HTTP via methods we've discussed, like GET and POST
- You can use this to read data, either through special URLs, or via GET or POST, and to create / update data (also via POST, but can be others)

Examples:

(these are referred to as **endpoints**)

- https://api.github.com/repos/jvilk/browserfs
    - jvilk = username
    - browserfs = repository
- https://api.github.com/repos/jvilk/browserfs/stargazers
    - lists the people who have "starred" the repo

Notice response is in JSON. **We will want you to implement REST APIs for your backend in your project.**

Fetch

"XHR"

Lets you talk to servers from your browser. It's pretty simple!

1. Request a URL (potentially with headers) from a server -- this **sends** info to a server -- and then you **await** the response.

```
const response = await fetch('https://some.fcking.url/');
```

You can add GET stuff into a URL if you want:

```
const response = await
fetch('https://get-persons-age.com?first_name=Sarita&last_name=Kumar');
```

To use POST instead (better, recommended), add **headers** (second argument to **fetch**).

```
const response = await fetch('https://getage.com', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify({ first_name : 'Sarita', last_name : 'Kumar' })
});
```

2. Make sure to check if the response succeeded!

```
if (response.ok) {
  // do something
  // ...
} else { // error occurred
  // the code is in response.status, if you care (e.g., 404)
}
```

3. Do something with the response to read the body. The web server could return HTML, JSON, raw text, or a **binary** "blob" (for images)

```
const someJSON = await response.json(); // now you have a JSON object
// or
const someHTML = await response.html(); // etc.
// or
const someBlob = await response.blob(); // see https://javascript.info/blob
```

^^ **You can only do ONE of these!**

4. Once you have the response object, you can then do something with the response.

{ 'age' : 23, 'zipcode' :'01002', }

```
const age = someJSON.age; // assuming the JSON object has age
```

Exercise!

COMPSCI 326 F20 - 12. REST & Fetch Exercise