Fetch

Browser XHR (XMLHttpRequest) to make service calls

- It was horrible
- Many libraries made to help (jquery, axios, etc)

Now we have `fetch()`!

- No need for those other libraries

Fetch returns a promise

```
const promise = fetch('/people');
promise.then( () => console.log('fetch complete') );
```

The promise resolves with a response object (google: MDN Response)

```
fetch('/people/')
  .then( response => console.log(response.status) );
```

The response object does NOT have the parsed body

If you are getting data, you want the body

The body has not been parsed for the response object

Call a method to parse the body (`.text()` or `.json()` for example)

These parsing methods **are async**

```
fetch('/people/')
  .then( response => response.json() )
  .then( body => console.log(body) );
```

Using the body

```html
<ul class="example"></ul>
```

```javascript
const list = document.querySelector('.example');
fetch('/people/')
  .then( response => response.json() )
  .then( people => {
    const names = people.map(
      name => `<li>${name}</li>`
    ).join('')
    list.innerHTML = names;
  });
```

But: This updates the DOM directly - bad idea!

What is better?

Better design

```html
<ul class="example"></ul>
```

```javascript
let names = [];

const list = document.querySelector('.example');

fetch('/people/')
.then( response => response.json() )
.then( people => {
  names = people; // update state
  render();
});

function render() {
    list.innerHTML = names.map(
      name => `<li>${name}</li>`
    ).join('')
}
```

Why better?

state is maintained in variables

- not just in the current DOM
- can rebuild DOM at any time from state
- can consult state without checking DOM
- Keeps state management simple
    - unimpacted by changes to the DOM

We can change state in many places

- always call `render()` or `renderSomeSection()`
    - "render" is my name, but common concept

Handling errors

`fetch` promise is **NOT** rejected (error) when service returns an error

Service errors are successful communication

- Only network errors will be caught by `catch()`

Instead, you can check the status code

- Services with good status codes are important
- `response.ok` is shorthand for status code ranges

Is service error message in same format as success?

- (e.g. JSON)

Error Tips

- Don't leave the user confused
- console.log() is **NOT** error handling
- You almost never SHOW the error message directly from the service

Students lose multiple points on their assignments and projects every semester

- Tell the user what they need to do just like you see on actual websites

Error example

```html
<ul class="example"></ul>
<div class="status"></div>
```

```javascript
const status = document.querySelector('.status');
fetch('/people/')
.then( response => {
  if(response.ok) { return response.json(); }
  // This example service sends JSON error bodies
  return response.json().then(err => Promise.reject(err) );
})
.then( people => {
  const names = people;
  render();
})
.catch( err => status.innerText = err.error );
```

What about network errors?

Network Errors

```
const status = document.querySelector('.status');

fetch('/people/')
.catch( () => Promise.reject({ error: 'network-error'}) )
.then( response => {
  if(response.ok) { return response.json(); }
  // This example service sends JSON error bodies
  return response.json().then(err => Promise.reject(err) );
})
.then( people => {
  const names = people;
  render();
})
.catch( err => updateStatus(err.error) );
```

Error Messages

- Don't leave user confused
- Rarely show direct messages from server
    - Often a code or key: use for your own messages

```
function updateStatus( err ) {
  const messages = {
    'network-error': "There is a problem connecting to the network, please try again
    'not-enough-catnip': "Your request did not contain enough catnip.  Please correc
    'nap-time': "Services are unavailable due to it being nap time, please try again
    'default': "Something went wrong!  Please try again later",
  };
  const status = document.querySelector('.status');
  status.innerText = messages[err] || messages.default;
}
```

Different methods

`fetch()` defaults to GET method.

It accepts an optional object

- The `method` key allows you to set the method

```
fetch('/people/', {
  method: 'POST'
})
```

Sending Data

Query params are sent as part of the URL

- the first argument to `fetch()`

Body params can be sent as the `body` option

- Remember: Not with GET
- Body params can be in multiple formats

```
fetch('/people/', {
  method: 'POST',
  body: JSON.stringify({ name: 'bob', age: 32 })
})
```

Sending Headers

There is a `Headers()` object and a `headers` property

```
fetch('/people/', {
  method: 'POST',
  headers: new Headers({
    'content-type': 'application/json'
  }),
  body: JSON.stringify({ name: 'bob', age: 32 })
})
```

NodeJS node-fetch has no `Headers()` - just pass an object

Consume a REST Service

- Page has empty `<ul class="posts">`
- On page load, populate `<ul>` with posts
- Make each post title a link
- Click on title:
  - hide other posts
  - populate and show a ul of comments
  - how to return to main listing?
- Show an error if an error
- **https://jsonplaceholder.typicode.com/posts**
- **https://jsonplaceholder.typicode.com/posts/1/comments**