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
- Polyfills available (remember polyfills?)

Node does not have fetch built in

- `node-fetch` package is available

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) );
```

Response object does NOT have 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

- Example: `.text()` or `.json()`

These parsing methods **are async**

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

Using the body

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

```
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?

- Updating state, then rendering

Updating state

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

```
let names = [];
const list = document.querySelector('.example');
function render( names ) {
  const html = names.map(
    name => `<li>${name}</li>`
  ).join('');
  list.innerHTML = html;
}

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

- We will cover even better breakdowns soon

Handling errors

`fetch` promise **NOT** rejected if 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 example

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

```
const status = document.querySelector('.status');
fetch('/people/')
  .then( response => {
    if(response.ok) { return response.json(); }

    return response.json().then(err => Promise.reject(err) );
  })
  .then( people => {
    const names = people.map(
      name => `<li>${name}</li>`
    ).join('')
    document.querySelector('.example').innerHTML = names;
  })
  .catch( err => status.innerText = err.error );
```

What about network errors?

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

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

Credentials

By default, `fetch()` won't send cookies on requests

- You alter this by passing an option
- `credentials: 'include'` in fetch() options object

Useful if a service call makes use of cookies

Confirm a list of names REST service

- Static HTML page has an empty `<ul></ul>`
- `GET /people` on page load
    - populate the list of names
- Put an `x` next to each name
    - call `DELETE /people/:name` when clicked
    - update state and rerender the list of names

Next:

- Show errors to user in HTML
- Translate error codes to nice messages
- Add new names