Using REST APIs

Michael Chang

Spring 2020

Schedule update

assign3 out tonight

Due next Wednesday

assign4 should go out right after assign3 due

Due Wednesday week 8

Project proposal feedback as soon as we can

Project check-in / milestone

Single formal check-in, info soon

Plan for today

Recap: fetch with async/await

- Handling errors

Using REST APIs

- Receiving data
- Sending data: method, headers, body
- Classes to represent data

async/await gotchas

Can't use await in non-async function

If you make a callback that uses await, it has to be async too

```
const main = () => {
  let elem = ...;
  elem.addEventListener("click",
  async (event) => {
    let res = await fetch(...);
    ...
  });
};
```

async/await gotchas

async functions return Promises

Even if you don't use await

```
const foo = async () => {
  return 42;
};

/* Can mix/match async and Promise.then */
foo().then(num => {
  console.log(num); // -> 42
});
```

async/await gotchas

If you leave off await, bad things happen

You'll get a Promise, which is probably not what you want

```
const foo = async () => {
  let response = fetch(...); // No await!!
  let text = response.text();
  // Error: Promise has no text() method
};
```

Unfortunately, this can be really hard to debug

Aside: exceptions

try/catch blocks

```
try {
  ...
  throw new Error("Boom");
  ...
} catch (e) {
  console.log(e.stack);
}
```

Aside: exceptions

`throw <expression>`

- Can technically throw anything
- But probably should throw `Errors`

`new Error(message)`

- Automatically builds a stack trace
- Displays nicely in the console
- Can have subclasses of errors

Sending data to server

`fetch(url[, options])`

- options is an object with following keys
  - method: HTTP method
  - headers: HTTP headers to include in request
  - body: request body (for non-GET)

When sending data to server

- Recall: query string goes in the URL
- When including request body, need to set Content-Type header

Sending data to server

```
const postData = await () => {
  let data = { num: 42 };
  let res = await fetch(
    "/api/path?param=binky",
    { method: "POST",
      headers: {
        "Content-Type": "application/json"
       },
       body: JSON.stringify(data)
     }
   };
   ...
};
```

Data models

Useful to encapsulate data in classes

E.g. Student, Course

Methods of reading and updating from API

Note: constructor cannot be async

Instead, use a `static` method

Data models

```
class Student {
  static async load(id) {
    let res = await fetch(...);
    let data = await res.json();
    return new Student(data);
  }
  constructor(data) {
    ...
  }
}
let student = await Student.load("mchang");
```

Data models

Useful function: Object.assign(dest, src)

- Copy all the keys from src into dest (overwriting)
- E.g. `Object.assign(this, data)`

`myClass.toJSON()`

- Define this method to control how `JSON.stringify` converts object into JSON
  - E.g. include only public instance variables

Summary

Today

- Done with first pass client side (really this time)

Next time

- Start talking about servers
- Writing these APIs

Next week

- Storing data