Fetching Data

Web Dev, Spring 2021

Recap

JavaScript to manipulate the DOM in response to events

MVC as one way in which to structure code

We basically have all the ingredients to build frontends: connect actions to events

One bit that's useful but missing: getting data from elsewhere

- we know how to get data from the user [control elements]
- what about getting data from another system?

Where do we get data?

Browsers know how to communicate over HTTP with a web server via a URL

Default browser behavior is to get documents via HTTP requests

- if it gets an HTML document, it renders it
- web servers can send back any document in response to an HTTP request
- pictures, strings, json objects, ...

JavaScript can use the browser's ability to request documents from a web server

- not to render them, but to use the documents programmatically

REST APIs

A set of URLs that are meant to return data over HTTP not to be rendered by a browser but to be consumed by code

- could be JavaScript in a browser
- could be some code running in any desktop app
- Python, Java, Go all have libraries to make HTTP requests to a web server

We'll come back to REST APIs later in the semester

Example: Open Library API

How do we make an HTTP request in JavaScript?

There are two kinds of HTTP requests (actually, 9...)

- GET requests — default one sent by the browser
- POST requests — can pass extensive information to the server

We'll concentrate on GET requests for now

HTTP requests from JavaScript

- modern way: `fetch`
- old way: `XMLHttpRequest`

Fetch API

```
fetch(url)
```

- by default, sends a GET HTTP request to the given URL
- it returns *immediately* as soon as it makes the call
  - no, it doesn't wait for the response
- the result of the call is a *promise object*

Promises

A promise is an object that represents a value that hasn't been computed yet

> “*this is going to be a value some day, I promise*”

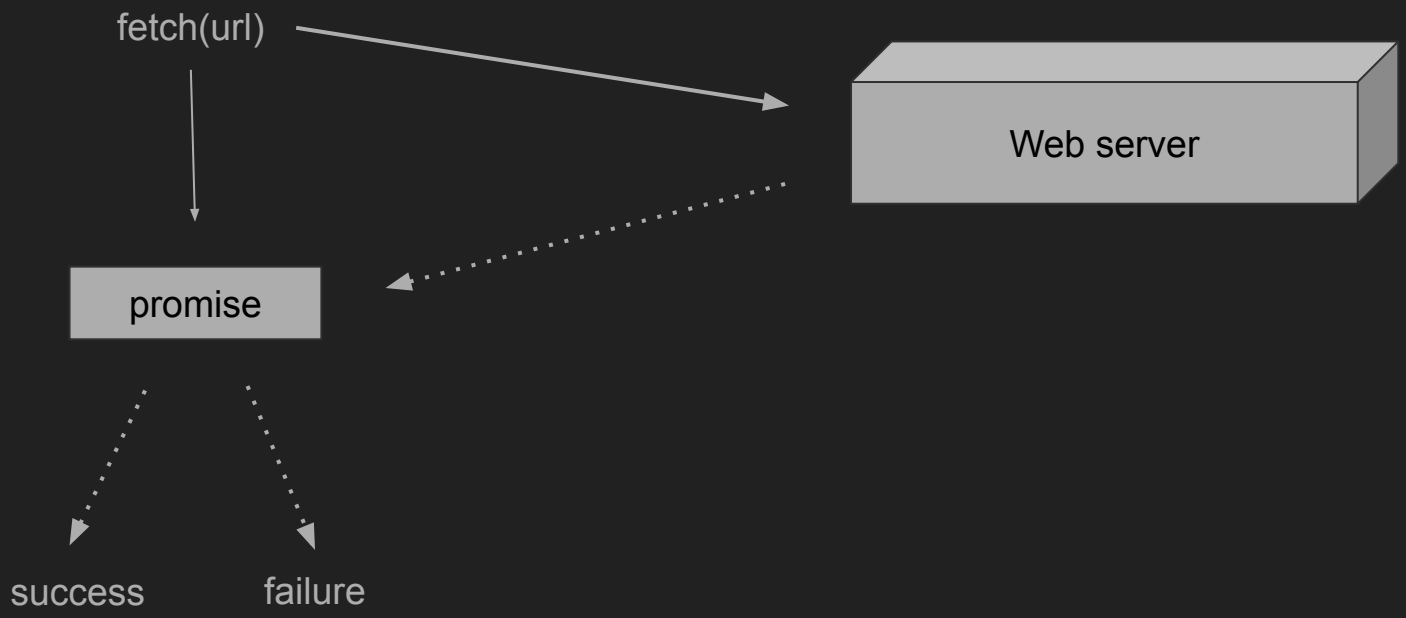
Computation of the value represented by the promise happens in the background

The computation eventually completes (or fails), and the promise calls a handler

- function you associate with a promise that says what to do when the value is finally computed (or the computation fails)

```
promise.then(handleSuccess, handleFailure)
```

Back to fetch...



Back to fetch...

`fetch(`*`url`*`)` returns a promise — associate a handler to process the response

```
const fProm = fetch(url)
const hSuccess = response => do something with response
const hFailure = err => do something with error
fProm.then(hSuccess, hFailure)
```

(method `.then()` returns a promise that represents the value returned by the then success handler (if any) — meaning you can chain `.then()` methods to get a cascade of promises computing results based on earlier in the chain.)

What sort of things do we do in a fetch handler?

We don't return a value from the fetch handler (where are you returning to?)

We send the value somewhere else

In particular — we can invoke an action on a model to change model state!

- it's very much like a user action, except it's a programmatic action

This is where something like MVC is useful

Demo — adding fetch from picsum