Service Workers and IndexedDB

CS-554 – WEB PROGRAMMING

IndexedDB: Like localStorage, but not.

What is IndexedDB?

IndexedDB is a data store that exists in your browser that allows you to store documents across page loads. It is, perhaps, the most powerful way of storing data on your users browser.

- Can index fields for quick querying
- Can version databases
- Can use on workers and on main window!

The API is a little raw; you'll probably want to abstract it out a little bit if using it extensively

How do we we use IndexedDB?

The usage of IndexedDB requires a little more setup than using localStorage

- You first make a connection to a particular database; you can have multiple databases per app
- You setup your event handlers to define the schema
  - Each database has many “stores” that you can add (think MongoDB collections); you define the “id” of each store
  - You can add indexes on fields in that store
- You setup event handlers for what to do when DB is open

The IndexedDB api is dumb and doesn’t use promises; as usual, the open source world saves us and gives us libraries to handle that if we need them

- https://github.com/jakearchibald/idb
- Fairly easy to abstract most simple operations using the native Promise API anyways
- We’ll use that in real situations, because we’re spoiled and don’t program without async / await; it’s not 2001, we deserve better than that

Example

Loading up this week's codebase, we will make a simple web server that acts as a to-do list and utilizes a service worker.

Our service worker will utilize IndexDB to store to-do list entries, in the event that our POST call fails

Service Workers

HTTP GET /HYPE

Service workers are one of those technologies that are currently very, very hyped up and heralded as the *next big thing*™

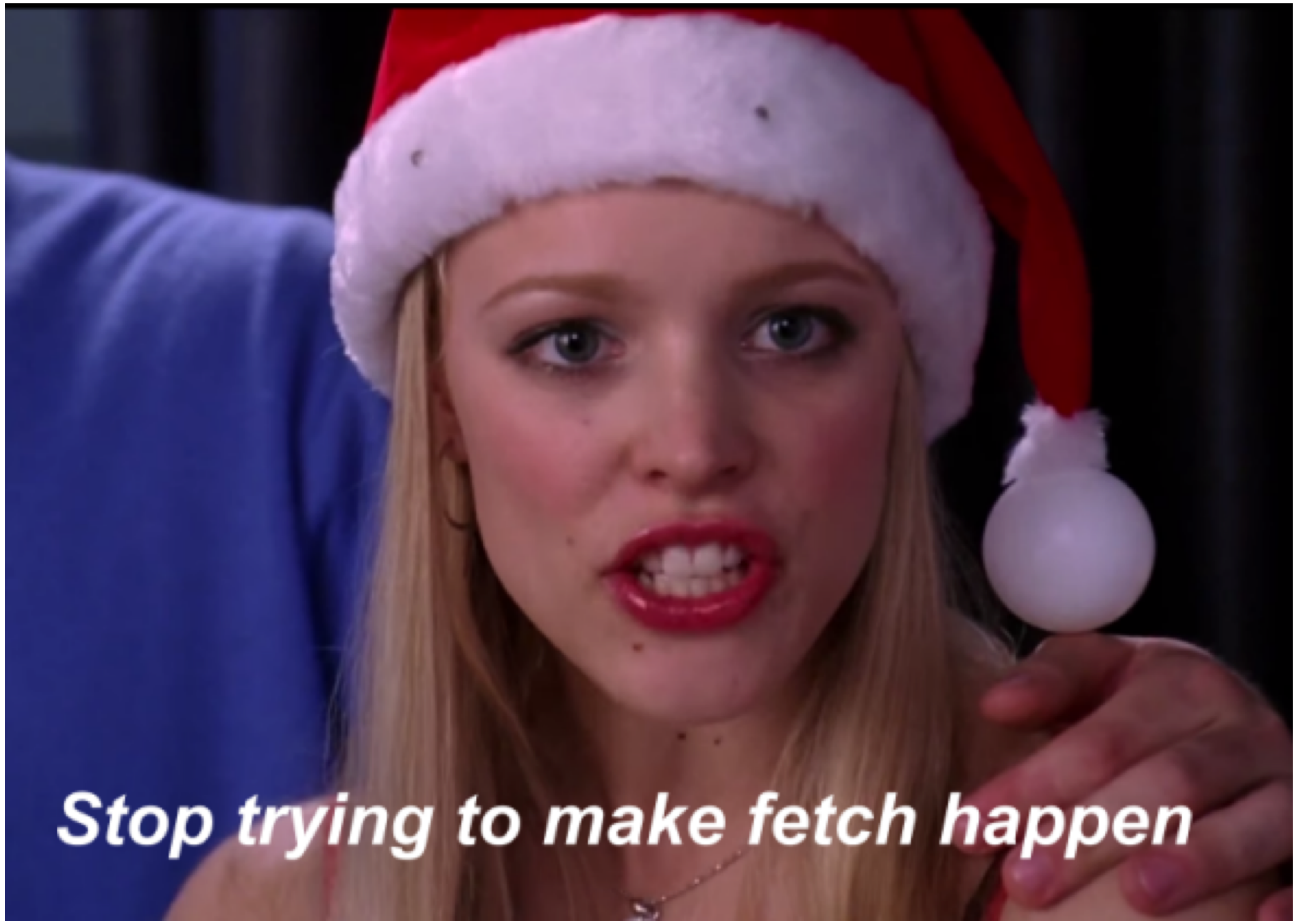
Service workers are workers that can intercept network requests, cache, and handle some other application-level things, like push notifications and background syncing (coming soon, to a browser near you)

Service workers allow you to develop your app **offline first**; you can treat your network as optional, and something that you send your data to at a later point in time

Service Worker Life Cycle

The Service Worker API is highly promised-based, and offers a simple life cycle:

- You register a service worker in your ”main” page (your DOM one)
- Your service worker is registered, and at some point will spin up; or maybe not, that’s cool, too. These are designed to go up and down repeatedly. We’re not dealing with one single instance of them.
- Our service worker will run, and fire off an install event
  - If that event is listened to, we can dictate an asynchronous install set of instructions; for example, making an indexed-db pool or
- Our service worker will fire an ‘activate’ event, when it’s ready for processing; we can use that to clear out old cache entries, or whatever
- We can then do the most important part, and intercept the **fetch** event.



Fetch

DESPITE REGINA GEORGE'S BEST EFFORTS, FETCH HAPPENED.

No, really, Fetch happened

Service workers have the ability to intercept network requests, by listening to the “fetch” event.

By listening to this event, service workers have the ability to analyze data about the request and manipulate it according:

- Determine whether or not the request should be served from the cache
- Determine whether or not the request should be cached upon completion
- Determine whether or not the request should be manually “responded to” with a response generated in the worker

Caching Assets

We can leverage a cache in our service worker, that allows us to cache responses; this means that we can store assets, like our actual application files, that are served **when we are offline**

Bam, we just made a progressive web-app.

Example

Fire up this codebase and see the world's most boring to-do list

- We start off simple, using Bootstrap 3 to render a beautiful page
- We register a service worker and our UI code never writes another line of service-worker logic again
- We make an HTTP request to /to-do items and render out the request using some basic, basic jQuery manipulation
- When we submit the form, we POST to the server and

While our service workers:

- Register, and setup the IndexedDB database schema
- Cache the jQuery, bootstrap assets in an **asset** cache
- Cache the GET /to-do route in an **appData** cache
- Check if POST /to-do fails, and throws the entry into IndexedDB for future uploading

https://github.com/Stevens-CS554/service-workers-indexeddb