Let's take this offline: IndexedDB

Posted in Lets-take-this-offline and tagged Offline-First , IndexedDB , Connectivity , Internet-Connection , JavaScript , Lets-take-this-offline , KCDC on Jul 17, 2018

Last week I was at KCDC and it was absolutely amazing. I was lucky enough to not only attend, but also to speak about something near and dear to my heart – Offline First techniques. This time I was talking about using IndexedDB.

I've written about the importance of offline availability here. Offline First offers a kind way to handle a user's loss of connection – something over which they have little to no control. And even more than that – implementing Offline First techniques can help people, which I've written about here.

Before we dive into any code, let's talk about IndexedDB's structure. I think IndexedDB is more similar to NoSQL, rather than traditional relational SQL. So let's delve into how IndexedDB data is stored.

Each web app can have multiple IndexedDB databases, which are made up of Object Stores. An Object Store is kind of like a table, but it doesn't use columns. It's a place to store Objects, which will probably look similar to a server response or ajax request. Later on, we'll look at some examples.

Within an Object Store, there are Indexes. An Index is a defined property within an Object Store. It's a little bit like a column, except that it doesn't

have a defined type, nor does it require a type.

Indexes have various properties which can be applied to them. A few common ones include `autoIncrement`, `unique`, and `keyPath`. When we apply `autoIncrement` to an Index, the Index will create something similar to an auto-incrementing ID in SQL. I mentioned before that Indexes don't have type requirements, but every rule needs an exception. When we use `autoIncrement` our Index will always be a number. However, we don't need to specify a type for this to happen, nor do we need to pass in a number, as IndexedDB is going to handle that for us.

Another handy property we can apply is `unique`. `unique` allows us to specify whether or not an Index value can be duplicated. So for instance, if we wanted a list of email addresses, we might say that the email Index would be unique. This would keep any duplicates from being added. In the event that someone tries to add an email that is already in our Object Store, the email (and associated record) won't be added. This helps protect the integrity of the data.

The last property we'll discuss here is `keyPath`. The `keyPath` property allows us to define a sort of "master" Index within our Object Store. It's a little similar to a primary key in a SQL table, but like most Indexes, does not require a specified type. Defining an Index as a `keyPath` allows us to look up records directly by that property.

Using our email example from before, if we specified the email address Index as our Object Stores `keyPath`, we could look up "ceo@bigdealcompany.com" directly, and get that record. If the `keyPath` for a record were a name instead, we would need to look up "Patricia CEO" to find the email address. Specifically, we cannot directly look up records via an Index that *has not* been defined as a `keyPath`.

Now that we have a good idea of IndexedDB's structure, we'll look at some code in the next post.

Questions? Tweet me!

MargieMap: Mapping The Internet

Posted in Lets-take-this-offline and tagged Offline-First , Lets-Take-This-Offline , Mapping , Offline-Access , Intermittent-Connectivity on May 18, 2018

Earlier today I was lucky enough to speak at 200OK in Tulsa. My talk, Mapping The Internet, is about a project I've been working on called MargieMap.

MargieMap is a visualization of United States census tracts by income. It also layers libraries on top of the census tracts. I used Mapbox to host the map tiles. The income data comes from the U.S. Census Bureau and the library data came from the Institute of Museum and Library Services.

I'm planning to open source the data I've gathered, but I haven't found a suitable (read: affordable) way to do that yet.

Why income?

I've been interested in the correlation between income and internet access for a long time. A lack of Internet Access contributes to the Homework Gap.

Of course a lack of internet access isn't limited to households with school-aged children, nor are it's challenges. According to a 2015 Pew Research Study nearly 20 percent of adults in the United States are smartphone-dependent – meaning they do not have access to the Internet outside of their smartphone.

These individuals reported using their phone for a multitude of things including job searches and looking up medical conditions. A large number of these individuals also reported losing their cell phone data due to cost.

With internet access being so prevalent, it's hard for some developers to imagine not having connection, but the truth is – everyone will lose connection. Losing connection is something that is almost completely

beyond a person's control. Unfortunately, it's usually beyond the control of the person when they'll regain their connection.

Why libraries?

Free wifi is a very common feature for restaurants and coffee shops, but the truth is that wifi isn't really free. Patrons of the establishment are expected to buy things in order to hang out and use the wifi.

Public libraries are one of the few places a person can go and use the internet without having to pay for a meal or a drink – which is really important if someone is unemployed and searching for jobs online. There are so many reasons why someone might need to use the internet – to research an illness, pay bills, or research for school. Fortunately, public libraries offer internet access without a fee.

How can I use the map?

The map has categorized incomes by color – low incomes are heavy blue and fade as the median income increases. Higher incomes start off light and turn into a heavy pink. You can search by zip code, city, or state and see the incomes in that area. If you click on an area on the map, you'll see the median individual income for that tract.

You might also notice orange dots scattered around the map – those are public libraries. The goal of the map was to visualize the income data along with libraries and see if there's any correlation between income and libraries.

What's next?

I've gotten data for all 50 states onto the map, but I also have data for U.S. territories as well. I plan to map the income data for the territories, but I've been struggling to find a list of libraries for the territories. If you know where I could find this data, please let me know!

Starting small, I'd like to find high/low incomes for the country, and for each state as well. Once I have that, I'll be able to focus on those income tracts and see if they have libraries and how many.

There's so much I want to do with this data – MargieMap is just the first step. I'm very excited to see what all I learn, and I'll be sure to blog about it too!

*** I would be remiss if I didn't mention all the help I got from Devin Clark. Devin is a JavaScript dev with a specific interest in geo and mapping, and he was a great resource on mapping tools. Thanks Devin! ***

How to setup HTTPS on Github Pages

Posted in Maintenance and tagged Website , Blog , Github , Maintenance on May 1, 2018

I've been considering moving off of Github Pages for a while because of issues with HTTPS not working with custom domains.

But Github Pages makes everything so easy! Just push to the repository where my code would live anyway and boom, it's done. So when they introduced HTTPS for custom domains, I was ecstatic!

The only problem was, I couldn't find specific steps all in one place for how to do it. So I wrote down what I did here so you don't have to have six tabs open like I did.

1. If you see an error about not being able to use HTTPS, check your DNS provider.

You probably need to change the IP address your domain's A record is pointed to. A list of IP address you can use is available here.

2. Go into your repos settings and delete the custom URL and save.

2.1 While still on this page, re-add the custom URL and save.

This will trigger the CNAME file being deleted from your repo, and then re-added which will trigger the HTTPS availability.

3. While still on the settings page, click `Enable HTTPS` underneath your custom url.

This took a couple of minutes for me (less than 5 minutes) because they had to generate the certificate. Once the certificate is generated, you'll be about to click the checkbox.

Aaaaand, voila! You now have HTTPS enabled on your site.

Vive la Github Pages!

Questions? Tweet me!

What can I do when I'm stumped?

Posted in JavaScript and tagged JavaScript , Beginners , Learning , How-To , Getting-Started , Mentorship , Mentor on Apr 17, 2018

A pretty typical beginner dilemma: you're completely stumped by an issue with your code. You've tried multiple solutions, but nothing is working. You're stumped.

But, you're also worried about reaching out for help because you don't want to reach out too often. What can you do?

Well, if you're in this situation, I hope firstly that you have someone more experienced you can reach out to. That helps tremendously. Mentors can be found at local usergroups (you can check meetup to find usergroups in your area) or even online using coding forums.

If you've already secured a mentor, you're ahead of the curve! But if you're worried about asking too many questions, it can be tough to know when

you should keep working on a problem versus when it's been long enough and you should ask.

Some people advocate a time-based system. Something like, "don't spend more than X hours on a problem." I propose another system – one that isn't time-based.

A system that has worked for me in the past is to try three things before asking any question. It's important to also write down what you're trying and the result you're getting.

This has multiple benefits. Firstly it gives you an exact template to use if you aren't sure how to ask your question. You can simply tell your mentor that you tried X and then Y happened, but you were hoping for Z outcome. This will help your mentor because they can review what you've been attempting and put themselves into your mindset. They'll understand better where your confusion lies and be able to explain the issue – and hopefully the solution – in a way you'll understand.

The other benefit is, in my opinion, even more important and beneficial. I've found this method very helpful because usually the act of trying something and documenting the outcome will give you something to search for online. That, in turn, will point you in another direction – quite possibly the direction you need to go! Oftentimes I didn't even need to ask for help because this method would lead me to an answer, or at the very least a different problem. Often when you're stuck on one problem, getting a different error will feel like progress!

One last point I want to make is how important your mentor is. A good mentor will be able to tell you to take a break and come back to the issue or tell you to keep working on a problem because you're close. They won't belittle your attempts or make you feel stupid because you didn't come to the answer on your own.

It's perfectly fine, even encouraged, to be conscientious of someone else's time. However if you feel like you can't approach someone because they make you feel less than, go find a new mentor.

I hope this helps. If you have questions, feel free to tweet me!

Let's take this offline: Intro to Service Workers

Posted in Lets-take-this-offline and tagged Offline-First , Service-Workers , Connectivity , Internet-Connection , JavaScript , Lets-take-this-offline on Dec 19, 2017

Service Workers are a super cool bit of tech that allows your user's browser to cache files and assets. Why is this so cool? Because eventually your user will lose connection. What happens then? Without something to bridge the gap between the time user loses connection and the time they regain connection, your user is just out of luck.

But no more! Service workers are here to help bridge the gap (at least in certain browsers). With the browser caching the files and images you choose, your user can continue using your site until the connection comes back.

Note: Service workers on their own cannot cache AJAX requests. I'll talk about solutions for caching API requests in a later blog post.

To start using a service worker you'll need some JavaScript for your application, as well a service worker file. I've created a demo Let's get started!

Step One: Register your service worker

Registering your service work is very simple. The code looks like this:

```
if(navigator.serviceWorker) {
  navigator.serviceWorker.register('service-worker.js');
```

```
}
```

The above code would go in your application's JavaScript file. All it's doing is letting the browser know to go ahead and register that service worker. Be sure to wrap your register function in an if-statement because cross-browser support for service workers isn't quite there yet.

Step Two: Install your service worker

This code lives in your `service-worker.js` file:

```
// service worker
self.oninstall = function() {
  caches.open('cash').then(function(cache) {
    cache.addAll([
      'index.html',
      '/img/my-image-name.jpg'
    ]);
  });
};
```

Service workers expose certain events, including `oninstall` and `fetch`. The method above opens a cache in the browser and adds files to the cache. This way, if your app goes offline, the browser already has files and can load them in. Another benefit of this is speeding up your site's response time. Since the files are cached, the browser no longer needs to serve files from a remote server. This ends up giving your users a great performance increase.

Step Three: Specify what happens on fetch

This code also lives in your `service-worker.js` file:

```
// fetch function
// called when connection is lost
self.onfetch = function(event) {
  // this will grab the very last request
  event.respondWith(caches.match(event.request));
};
```

This is the function that will be called when your files need to be fetched. When files are cached, they are saved like a stack. Older caches are closer to the bottom, with newest caches being on top. The above function will grab the latest cache and send that to the browser where it can be rendered.

And with that, you're done! You can grab this code and run it in a server (I like to use https-server). Once you stop the server, you can refresh the page and see your site is still there! Pretty cool, huh?

Questions? Tweet me!

Older →

I'm Carmalou, JavaScript extraordinaire!

Not really, but I am really good at Googling things that I don't know!

I made this blog to share the things I've learned with others, so if you have questions, drop me a line!

You can contact me via: Email / Github / Twitter

Copyright Notice

Recent Posts

Categories

Learning (1)
Lets-take-this-offline (5)
Live-coding (1)
Mac-Magic (1)
Maintenance (1)
Offline-First (1)
Projects (1)
Raspberry-Pi (1)
Series (1)
Speaking (3)
Weird-Nuances (3)

Tags

AJAX ASP.NET AngularJS Appcamp Apps Ask-Questions Async Automation Beginners Better-Alert Blog Bots C# C-Sharp Callbacks Camp Carver CoffeeScript Command-Line Complete-Guide Complete-Guide, Conferences Connectivity Console.log Databases Debugging Demos Development Dynamic Electron Environment-Variables Falsy-Values For-Loops Functions Games Get-Started Getting-Started GitHub Github Hair-loss Hemingway Heroku Hoisting How-To How-To, Hubot Hybrid-Apps IndexedDB Intermittent-Connectivity Internet-Connection Internet-of-Things Interviews IoT IoT, Ionic JavaScript KCDC Learning Lets-Take-This-Offline Lets-take-this-offline Libraries Literature Live-coding Loops Loosely-typed MVC Maintenance Mapping Mentor Mentorship Mobile Mobile Mobile-Development Native-alerts Navigator Nervousness Node Offline-Access Offline-Apps Offline-Camp Offline-First Offline-first Phaser Planning Plugins Projects Promises Public-Speaking Public-speaking Raspberry-Pi Raspberry-Pi, React Reference-Types SQL Security Series Service-Workers Shakespeare Speaking Star-Wars Terminal Troubleshooting Truthy-Values Twitter Vanilla-js Website Weird-Stuff hostname jQuery raspian raspian, ssh ssh,

Archives

2018

2017

2016

2015
