Accounta-Billie Part 1: Introducing Accounta-Billie

Posted in Accounta-Billie and tagged Side-Projects , Progressive-Web-Apps , Accounta-Billie , JavaScript on Aug 28, 2020

I wanted to take some time to write about a project I've been working on called Accounta-Billie. Accounta-Billie is a productivity and goals app. The concept is simple: you and your friends make up a group within the app. When one person in the group creates a task, such as "complete Accounta-Billie V1", the whole group can see it. Once you've made this commitment to complete your task, your friends can help hold you accountable, and you can help hold your friends accountable to their goals as well.

It may seem counter-intuitive to try to make big goals (like building an app) during this time of COVID. I've seen tweets about how much people have "gotten done" during this time, and other tweets imploring people to "use this time wisely". I find that kind of gross. Life is so much more than just checking items off a checklist. During a time like this, I really think we should just let everyone be. There's no need to use this time to complete all of your life's accomplishments if you're having trouble right now - or any time for that matter! Life is too short to spend your time doing anything that you don't want to be doing.

If you're wondering how I got from the previous paragraph to "Let's build an accountability app," let me tell you a bit about my own quarantine experience. I started quarantine thinking, like a lot of people, that this wouldn't last too long. I can remember thinking back in March that I only had to get to the summer, and then my kids would be in daycare, and I could I get some work done and also have my sanity back. And then when summer came and cases were worse, we made the decision to keep them home. I told myself that I only needed to get through the summer, and once school started back up, everything would go back to normal. Since then, the tough decision was made to keep my kids home and do virtual school. And even since that decision was made, my kids' school district has decided not to have in-person classes at all.

During this never-ending (and also somehow always changing) quarantine, I adopted some pretty unhealthy habits, with the idea that I was just coping with a short-term situation. I justified it, telling myself that it was only until, only until, only until.... But the habits weren't the kinds of habits that even made you feel better. Sure, they might have helped in the short term, but, if anything, they just compounded the depressed feelings even more in the long term. So once we

decided to keep my kids out of school, I knew something had to change. In that moment, I suddenly snapped out of my short term thinking and started to accept this situation as a long term sort of "new normal."

I wanted these new habits to start small, so my goal was to begin exercising regularly (of course, I didn't know then what "regularly" meant to me), in hopes that exercise would help lift my spirits. I mentioned this to my sister, and she was really excited about the idea of us being accountability buddies. This made a lot of sense, both to help us each stick with our goals, but also because right now, we can't see each other. We hoped that this would help keep us in contact.

But we couldn't find an app that was quite right. She suggested the typical fitness apps, but they didn't allow you to share tasks. I suggested a spreadsheet, but she said she probably wouldn't use anything that she couldn't access on her phone (she also called me lame for suggesting a spreadsheet). Since we couldn't agree on what to use, naturally, the next step was to begin building my own app. I'll detail everything over the next few blog posts, so stay tuned!

I hope you'll check out Accounta-Billie. I'm looking for people for the beta, which is launching really soon. If you head on over to accountabillie.com, you can sign up there. I would love for you to use it, check it out, and then to get your honest feedback. Check back soon for my post about my first weekend working on Accounta-Billie and finally conquering auth.

Let me know what you think on Twitter!

How to manage navigation in Xamarin

Posted in Xamarin and tagged C# , C-Sharp , Xamarin , Mobile-development , Mobile , Hamburger-menu , Navigation on Apr 20, 2020

As someone who loves JavaScript as much as I do, I never expected to be working in Xamarin. But that's where I've been since I started my awesome new job! Xamarin brings some interesting challenges. You might think it's just regular old C#, but since Xamarin uses MVVM architecture versus MVC architecture, I found a few things very tricky to manage. In this blog post, we'll talk about how I managed navigation in a master detail app.

So in this app we used a master-detail app, which included a hamburger menu. The master property contained the hamburger menu, while the detail contained whichever partial view page was meant to be the current content. The code behind included a built-in `Navigation` property. You can use it like this:

```
await Navigation.PushAsync(new MyPage());
```

According to Xamarin docs, most or all of our app's logic should be moved to the ViewModel. You can access the navigational properties via `App.Current.MainPage.Navigation` and push pages to the stack, as well as remove pages. However, I ran into a problem where the hamburger menu was disappearing when I would use `App.Current.MainPage.Navigation`. I realized I needed to

find a way to actually update the Detail property itself. Unfortunately the Detail property doesn't seem to be easily accessible. So I added a new constructor to the MainPage like this:

```
// MainPage.xaml.cs
// MainPage constructor
public MainPage(string page, params object[] args) {
    Type myPage = Type.GetType(page);
    Detail = new NavigationPage((Page)Activator.CreateInstance(myPage, args));
}

// FlowerPage.xaml.cs
// call MainPage page and pass in page, and any parameters
string type = typeof(FlowerDetailPage).AssemblyQualifiedName;
List<string> params = new List<string>();
params.Add("I am param");

App.Current.MainPage.Navigation.PushAsync(new MainPage(), params.ToArray());
```

This works well enough, however it seems a little yucky to keep adding new MainPages to the navigation stack. It seems like it would be better to directly update the Detail property, but how can we access it?

We ended up creating a navigational client with a method like this:

```
// NavigationClientManager.cs
public static async Task SetDetail(Type page, param object[] args) {
    MainPage mpage = ((MainPage)((NavigationPage)App.Current.MainPage).CurrentPage
    Page detailPage = (Page)Activator.CreateInstance(page, args);
    mpage.Detail.Navigation.PushAsync(detailPage);
}

// FlowerPage.xaml.cs
// set the detail page
List<string> params = new List<string>();
params.Add("I am param");

await NavigationClientManager.SetDetail(typeof(FlowerDetailPage), params.ToArray()
```

This allows the ViewModel to simply call the `SetDetail` method, and avoid all of the casting of types in the nav client. It really cleans up the ViewModels and makes it very clear what is going on.

If you have another way of managing navigation in a master-detail app, I would love to hear it. Let me know what you think!

Shout out to my friend Jacob Danks who helped me come to this solution. Thanks Jacob!

Caches: Caches.match

Posted in Lets-take-this-offline and tagged Offline-First , Service-worker , Connectivity , Internet-Connection , JavaScript , Lets-take-this-offline , Cache-Storage , Cache-API , Caches.match on May 1, 2019

Thanks so much for reading! If you want to learn everything you need to know about service workers, head over to serviceworkerbook.com and order my book, "Let's Take This Offline"! Be sure to use `caches-match` for 10% off!

Arguably, the biggest part of service workers is being able to fetch data off of a user's machine, rather than making trips across the network. There's a variety of reasons reaching over the network isn't ideal – lack of connection, intermittent connection, speed of the network. Fortunately service workers help with all of these things.

In this post, we'll look at `caches.match` – the function to fetch data from a service worker.

There are two matching methods within the Cache API – `caches.match` and `cache.matchAll`.

While `caches.match` requires at least one parameter, `cache.matchAll` has no required parameters. First, let's dig into `cache.matchAll`.

`cache.matchAll` can only be run on a single cache. It has two optional parameters – request and options.

```
(async function() {
    var cache = await caches.open(cacheName);
    var rez = await cache.matchAll(request);
}())
```

If the cache contained a key-value pair matching that request, the associated response would be returned. If there were no match, `rez` will be undefined.

The steps look like this:

1. Check that the request exists as a key in the list

2. If it does, add it to a response list

3. Return list of responses

Source: Service worker spec

You can call `cache.matchAll` without a request, and it will return *all* the responses in the specified cache.

So what's the difference between `caches.match` and `cache.matchAll`?

`caches.match` must be run with a request parameter. Additionally it'll check over each and every cache in the app. It might look like this:

```
(async function() {
    var rez = await caches.match(request);
}())
```

Unfortunately checking over each cache can result in it being a little slow, especially if there are multiple caches created for an app. It can be better to specify a cache and search it specifically, or even only have only a single cache. That would look really similar to our code up above with `matchAll`.

```
(async function() {
    var cache = await caches.open(cacheName);
    var rez = await cache.match(request);
}())
```

Thanks so much for reading! If you liked this post, you should head on over to serviceworkerbook.com and buy my book! Be sure to use `caches-match` for 10% off!

Cache Storage: What's the difference between cache.add and cache.put?

Posted in Lets-take-this-offline and tagged Offline-First , Service-worker , Connectivity , Internet-Connection , JavaScript , Lets-take-this-offline , Cache-Storage , Cache-API on Apr 23, 2019

If you want to learn everything you need to know about service workers, head over to serviceworkerbook.com and order my book, "Let's Take This Offline"! Be sure to use `cache-add` for 10% off!

What is cache.add?

If you've ever looked at a service worker, you're probably familiar with this little bit of code from an `oninstall` function:

```
cache.addAll([ /* ... */ ]);
```

Or perhaps you've seen:

```
cache.add(/* ... */);
```

Fortunately the syntax is pretty self-explanatory on what these two functions are doing. They are adding items to a cache. In one case they add multiple files, and in another, only a single file or request.

What is cache.put

Another function for adding items to a cache is `cache.put`. This function has two parameters: the request *and* the response.

```
cache.put(request, response);
```

So besides the obvious difference, that `addAll` and `add` only accept a single parameter (though of varying types), what is the difference between `add` and `put`?

The difference between add and put

It's a trick question! As it turns out, `add` and `addAll` use `put` under the hood.

It works like this:

```
1. For each request in the array (or a single request in the case of `add`), c
2. Add the response to a key-value pair list, with the request as the identify
3. Cache the request _and_ response together (by using put)
```

Source: Service Worker Spec

When to use add and when to use put

Usually in a service worker install function, you'll see this:

```
// code: https://github.com/carmalou/cache-response-with-sw/blob/master/servicewor

self.oninstall = function() {
    caches.open(cacheName).then(function(cache) {
        cache.addAll([ /* ... */ ])
        .then(/* ... */)
        .catch(/* ... */)
    })
}
```

It makes a lot of sense to use `addAll` over `put` in this case, because the user has never visited our site before. `addAll` is going to go fetch those responses and add them to your cache automatically, so we can use it during once our install event and no longer need to go over the network to get our static assets like HTML files. Additionally the syntax is very clean, so its very clear to future-us what we're doing.

So, if we should use `addAll` in an install event, where should we use `put`?

Before we get into the `add` vs `put` debate, let's look at our fetch event.

```
self.onfetch = function(event) {
    event.respondWith(
        caches.match(event.request)
```

```
        .then(function(cachedFiles) {
            if(cachedFiles) {
                return cachedFiles;
            } else {
                // should I use cache.put or cache.add??
            }
          })
        )
    )
}
```

In the above code, we have a fetch event sitting between the client making the request and the network which will process the request. Our service worker will intercept our request and check for a match within our cache. *If* a match is found, the service worker will return the response. Otherwise, we'll fall into the else.

Let's first look at using `cache.add`:

```
/* ... */

if(cachedFiles) {
    return cachedFiles;
} else {
    return caches.open(cacheName)
    .then(function(cache) {
        return caches.add(request)
        .then(function() {
            return fetch(request)
        });
    })
}

/* ... */
```

Briefly, should no match be found, we'll fall into the else. Within the else, we'll go ahead and open up the cache we want to store our data in, and then use `cache.add` to store off the response. At the end, we'll go ahead and do a fetch over the network so the client can access the data.

But there's a problem with this: we'll end up needing to do two fetches! Because `caches.add` doesn't actually return the response to the request it runs, we still need to do an additional fetch to get the response back to the client. Doing two fetches for the same data is redundant, and fortunately `put` makes it unnecessary.

Let's take a look at how we might rewrite this with `put`:

```
// code: https://github.com/carmalou/cache-response-with-sw/blob/master/servicewor

/* ... */

if(cachedFiles) {
    return cachedFiles;
} else {
    return fetch(event.request)
    .then(function(response) {
        return caches.open(cacheName)
        .then(function(cache) {
            return cache.put(event.request, response.clone());
        })
        .then(function() {
            return response;
        })
    })
}

/* ... */
```

So we've sort of flipped the order of what we were doing before with `cache.add`. Instead of doing the fetch last, we go ahead and do it first. After that's completed, we have a reference to the response. Next we can go ahead and get a reference to our cache with `caches.open` and use `put` to cache both the request and the response.

Note: Keep your eyes peeled for a blog post about why we are cloning the response!

Once we've finished caching the data, we can go ahead and return the response back to the client! And then next time the client makes this request, a trip over the network will no longer be necessary!

So, in conclusion, you should use `cache.add` or `cache.addAll` in situations where you don't need that data to get back to the client – such as your install event. When you're caching new requests, such as API requests, it's better to use `cache.put` because it allows you to cache the data *and* send it back to the client with a single network request.

Thanks so much for reading! If you liked this post, you should head on over to serviceworkerbook.com and buy my book! Be sure to use `cache-add` for 10% off!

How to cache API requests with a service worker

Posted in Lets-take-this-offline and tagged Offline-First , Service-worker , Connectivity , Internet-Connection , JavaScript , Lets-take-this-offline on Apr 16, 2019

If you want to learn everything you need to know about service workers, head over to serviceworkerbook.com and order my book, "Let's Take This Offline"! Be sure to use `cache-api-requests` for 10% off!

See the source code for this post.

Lots of service worker posts (like this one I wrote for the David Walsh blog) show you enough of a service worker to get started. Usually, you're caching files. This is a great start! It improves your app's performance, and with 20% of Americans experiencing smartphone dependence, it's a great way to make sure users can access your app – regardless of their connection or network speed.

But what about requests, and specifically `GET` requests? A service worker, along with the Cache Storage API, can also cache your `GET` requests to avoid unnecessary trips over the network. Let's look at how we would do that.

Note: This post assumes a basic understanding of service workers. If you need a service worker explainer, check out this blog post.

Let's take a look at our `onfetch` function in our service worker. It currently looks like this:

```
self.onfetch = function(event) {
    event.respondWith(
        (async function() {
            var cache = await caches.open(cacheName);
            var cachedFiles = await cache.match(event.request);
            if(cachedFiles) {
                return cachedFiles;
            } else {
                return fetch(event.request);
            }
        }())
    );
}
```

Briefly, this function intercepts a fetch request and searches for a match in the cache. If a match isn't found, the request proceeds as usual. The problem here is that there's nothing to go ahead and cache *new* data as it's received. Let's change that.

Our new function looks like this:

```
/* ... */
else {
    var response = await fetch(event.request);
}
```

If this looks similar to what you're doing in your client code, that's because it is! Essentially, we recreate the original fetch, but this time within the service worker. Now we'll go ahead and cache

the response.

```
/* ... */
else {
    var response = await fetch(event.request);
    await cache.put(event.request, response.clone());
}
```

There's a couple of new things in here. First, we are using `cache.put` over `cache.add`. `cache.put` allows a key-value pair to be passed in, which will match the request to the appropriate response. You might also notice the `response.clone()`. The body of a response object can only be used once. This means, if you cache the response object, you'll be able to return it, but your client won't be able to access the body of the response. To be able to access your data we'll go ahead and make a clone of the response and cache that instead.

Lastly, you return the response. So the full `onfetch` function looks like this:

```
self.onfetch = function(event) {
    event.respondWith(
        (async function() {
           var cache = await caches.open(cacheName);
           var cachedFiles = await cache.match(event.request);
           if(cachedFiles) {
               return cachedFiles;
           } else {
               try {
                   var response = await fetch(event.request);
                   await cache.put(event.request, response.clone());
                   return response;
               } catch(e) { /* ... */ }
           }
        }())
    )
}
```

There you have it! Now you're ready to start dynamically caching API responses.

Thanks so much for reading! If you liked this post, you should head on over to serviceworkerbook.com and buy my book! Be sure to use `cache-api-requests` for 10% off!

Older →

About Me

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

Xamarin (1)

Tags

AJAX ASP.NET Accounta-Billie AngularJS Appcamp Apps Ask-Questions Async Automation Beginners Better-Alert Blog Bots C# C-Sharp Cache-API Cache-Storage Caches.match 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 Hamburger-menu 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 Mobile-development Native-alerts Navigation Navigator Nervousness Node Offline-Access Offline-Apps Offline-Camp Offline-First Offline-first Phaser Planning Plugins Progressive-Web-Apps Projects Promises Public-Speaking Public-speaking Raspberry-Pi Raspberry-Pi, React Reference-Types SQL Security Series Service-Workers Service-worker Shakespeare Side-Projects Speaking Star-Wars Terminal Troubleshooting Truthy-Values Twitter Vanilla-js Website Weird-Stuff Xamarin hostname jQuery raspian raspian, ssh ssh,

Archives

2020
2019
2018
2017
2016
2015