To really make a point about this topic, when we are developing our applications, we typically develop applications into the medium that we are most comfortable with. And we have this thing about other areas that need also great experience.

For example, when we write our applications, some are not accessible until we are incapacitated in some fashion and then we see how important it is to really make our applications accessible.

Same thing when we are adding mobile applications, we write them assuming we're going to have great bandwidth and find ourselves in a situation where we are constrained with bandwidth and see that hey, it's not working the way I intended.

This is why we have offline first. It's to have the same experience across all network areas.

To talk to us about offline first is Bradley Holt, a software engineer at IBM and developer advocate. He's going to take us through what it means to maintain that user experience across all areas that we are touching with our applications.

Without any further ado, I'd like to invite Bradley to take us through this.

Thank you very much, McArthur, and thank you for joining us for today's tech talk.

So you can find me on twitter as add Bradley hold and also on medium. As mark said I work for IBM as developer advocate. One of the things I do is write and speak about a number of different technologies, including of course offline first.

I also co organize an event called offline camp, a really fun event, really happy to be a part of this. It's an unconference where participants in the event drive the agenda.

Our goal with these events is to build a community around the offline first movement. So far we have gone to the cat skills of New York, central coast of California, Berlin and Oregon, and we should be announcing our next location hopefully soon. So I hope you will consider joining us for our next offline camp.

Before I tell you today about offline first in my words, I first want to share with you a short video from some offline camp participants who share their perspective on what offline first means to them.

Off line first is an approach. It is when I build an application, I make sure it works under the most constrained circumstances. In terms of offline first, it's the connectivity. Then treat everything else as a progress enhancement.
I have never been so fragile and volatile, so seems we want to be building good software is and software that works with developers.

What offline first means to me is the possibility to make web applications nowadays to be much more accessible to more people.

The web as a stack and world is incredibly vibrant and we know this is sort of democratization of knowledge and Democratization of the development of tools. It's really powerful stuff. But it all falls apart the minute that you use great connectivity.

When we have to be connected to bandwidth, latency is always there. Even in the best of first world countries, it's not a given.

With offline first we're bringing content and data closer to end users. We can do that through a number of different technologies including couch TV.

Cool thing about those open source technologies, they allow us to sync data from a number of different types of apps, for example, progressive web apps, native apps, desktop using things like electron, even internet of things apps.

Offline first as well as progressive web apps as well as database technology talking about the couch, these are giving developers open and free tools so they can use to make experiences that are top-notch. At the end of the day when you boil it all down, we're talking about user experience.

No matter what, you always want your application to work as quickly as possible and return the results that you need in the time that the client needs it to be. Offline first is a clear, you know, solution to that.

The application needs to work at all times. And if it doesn't, you will lose engagement and you will eventually lose revenue in whatever your organization does.

Anything more than 100 or 200 milliseconds is perceived as slow. Everything has to go faster. On top of that, big companies like Google and Amazon have studies that show for every certain milliseconds they reduce income and revenue by many millions of dollars.

If you want to build applications that has really steadily fast user experience, the best thing you can do is go offline first, which is use preferring local cache in memory cache or on disk cache, prefer that at all cost to the network.

It's a very long path between the intricacies between networking latency and user experience from a technology stack point of view. But that is the breadth that offline first kind of requires.

Offline first presents a number of unique challenges. For example, the number of people in the offline first community are working to experiment around design patterns. So for example how to communicate where the user's data is, on a device or synced to the server or in the process of being synced. So provide some confidence to the user that they understand what's happening with their data.
The reason developers I think like off line first, whenever we are out in the real world and we encounter something that doesn’t work, we have the software coding abilities to turn that anger into opportunity and go and build something that's better. That's really what this community is about.

I think we're all familiar with the frustration of being off line, whether we're losing connectivity for a short period of time or whether it's a longer period of losing connectivity. And in off line first, what we're talking about today, even though the name is off line first, it's actually not just about being off line. In fact it's about much more. I think one of the most important things and most interesting things about off line first is its ability to provide a reliable, fast and engaging user experience regardless of the level of network connectivity that your end user has.

Most exciting to think about off line first is the ability, I think this ability to provide a level of performance by providing zero latency access to content and data stored locally on the device. Since we don't have to do a round trip to the network to get content or data, or to read or write data, we're able to provide this just really fast user experience.

The great thing about off line first is that it makes our apps better, whether we're off line, whether we’re on line, or whether we’re software in between. I don’t know if anyone here on the call has heard the term live fi before, but that's where the device has a wireless connection by your wireless router doesn’t have a connection to the internet. So looks like you have an internet connection, your device is telling you it has an internet connection, but it is in fact lying to you about that.

I want to talk to you about how this off line first approach is relevant to a number of different types of applications. So whether you’re building a progressive web app, whether you're building a hybrid mobile app using a tool like Cordova or ionic, whether building a native app for IOS or other platform, or building a desk top app using a term like electron, this approach can help you provide a better user experience for your ends users, the end users of your application.

Our team has actually built a series of demo apps. Actually just a simple shopping step. And what we have done, we actually rebuilt this same demo app in a number of different technology stacks. So you can have an example of building an off line first app in your stack of choice.

So we have built these as progressive web apps, we built a hybrid mobile app, several hybrid mobile app versions, native mobile app versions, even a desktop version using electron.

It's a really simple app. Basically you can create shopping lists. Within a shopping list you can add items to that list. So for example I can create a groceries, camping supplies list, and within that list add mangos, oranges, pears, later add bananas, it's a simple app and demonstrates the ability to work off line and provide the user experience regardless of the level of connectivity.

Then basically syncing the data in the background when you do have an internet connection.

We have built a vanilla JS version, meaning complaining JavaScript version. We built a version that users polymer preact version, a preact version, UJS version, ember JS version. These are all progressive web apps. These six that I talked about so far.

We also have a react native, ionic, cordava version, to demonstrate the approach in a hybrid mobile app.
We also have a swift and colony implementation to demonstrate the approach in a native iOS or android app, and also an electron implementation to show how you can take this off line with desktop app.

We are actually working to package these implementations up as developer patterns on IBM code. So the polymer implementation has already been published and we’re working on polishings and publishing the other implementations as well.

The code for these are actually on GIT hub and you can find these today for all the implementations I just talked about.

First want to talk about progressive web apps. This is a relatively new way of building applications. It’s an approach that I’m particularly excited about.

If you’re not familiar with PWAs or progressive web apps, they are basically websites that are also apps. So PWA provides the reliable, fast and engaging user experience of a native mobile app, combined with discoverability of a web app.

So you can basically link to a PWA just like a website, can be index or genuine because it is basically just a website. It gives you I think the best of both worlds, business both a website and a mobile app at the same time.

There are a number of case studies documented on PWA stats.com that demonstrate the positive business empackets for companies that have invested in building PWAs. You will see things documented like decreased page load time, reduced time to interactive, increased revenue, improvements in reengagement. These are just a few of the business benefits that companies have seen who have adopted this PWA approach.

On kind of a little higher profile example you may have heard of is twitter light. This made a little bit of news last year. Twitter light is in fact a progressive web app. If you go to mobile.twitter.com you are actually using a progressive web app.

The intent is to make something that has pretty much all the same functionality as the native twitter app, but it’s URL addressable, works like a website. As I said before indexable by search engines, all that great stuff.

May not be immediate obvious when you go to twitterlight that it is in fact a progressive web app. That’s the point. It should work like a website but it's faster, more capabilities, and gains additional app like functionality as you continue to use it.

For example, you can install twitter lite to your home screen on an android device.

With this PWA twitter seen as 65 percent increase in pages per session, 75 percent increase in tweets and 20 percent increase in bounce rate. So just a few benefits seen from taking this approach.

For me, think one of the most interesting things and exciting benefits about the progressive web app approach is the value it can have in developing markets. If you think about how the next billion people are going to connect to the internet, they are likely going to be using the devices with fairly decent storage and C. PU capabilities, probably goings to be using a browser like chrome, but probably not internet connectivity, probably 2G or 3G with data caps.
For this market segment, off line first progressive web app approach makes a ton of sense.

There are a number of device APIs available on the web platform today. More and more that you can do with progressive web apps every day. For example, PWA can be installed to the home screen on your device. You can grant the PWA ability to send you push notifications, ability to access your camera, your microphone or other device resources. There are other normal APIs like request storage, payment request API, geolocation API, just a few of the capabilities in API enabling PWAs to work more like you would expect a native app to work.

One of the biggest web platform API developments recently is introduction of service workers. Basically allow you as a web developer to get fine grade control over caching of content and caching of assets. So the way this works is that you write a little JavaScript code that gets registered as a service worker. That service worker then intercepts requests under a certain scope, basically under a certain URL path.

Then any network requests that then go after the service working is registered, any network requests that go to that URL under scope will be instead routed through your service worker. You then have that opportunity to intercept those requests with your service worker. Instead of those requests going to the network, you can actually reply with a request, with a response, sorry, that's stored in your local cache.

This is what we call a cache first approach where you're preferring a local cache over the network.

Of course at some point you originally needed to make a request to the network to actually get that content. So there was an initial network request necessary. But you can ensure that subsequent requests are served out of cache, which gives you a very fast load time with very low latency.

Service workers are fairly widely adopted at this point. So they are fairly safe to use across browsers. The big holdout was safari, but great news is that safari has now adopted service workers as well. So you can actually use this pretty broadly across different browser implementations.

I want to walk you through a little code snippet that demonstrates how service workers work.

The code you're seeing on this screen is code that would be in a JavaScript file that would be loaded with your initial page load. So the first thing you want to do is actually make sure that the browser supports service workers. So what we're going to do is check for the service worker property in the global navigator object.

If the browser does support service worker, then we're going to register a new service worker. So what we do, we pass the URL for the JavaScript file that contains the service worker, in this case SW test SWJS, and also going to pass the scope for the service worker. So what the scope means is that requests under this particular path are going to be intercepted by the service worker.

Basically a proxy of sorts that we get to write JavaScript code for.

Additional code here for what happens if the registration succeeds or fails, but we don't need to cover that today.

Now in the actual service worker file itself, so this is the SW.JS file, we're going to add the event listener for the install event. This event will be fired when the first service worker is first installed by the web browser.
Then we want to wait until the event has finished, sorry, we want to have the event wait until the code below has finished. That’s what this event.wait until does, the install event won’t be considered done until the following code completes.

Then we’re going to open up a cache called V1.

Actually a reason why you want to version your caches, about you we’re not going to cover that today. It’s a little too detailed for what we want to get into in today’s talk, but there is more information out there on service workers that you can find out about.

So yeah, we open a v1 cache, then we’re going to precache a bunch of assets, so we’re going to add a bunch of different assets to our kind of. So we’re going to add our html, some css, some JavaScript, a few images that we want to have precached after we first install the service worker.

Cool, that covers the install event for the service worker.

Then we also want to listen for the fetch event. So this is also in the service worker file. So also in the SW.JS file.

So basically whenever there’s an actual request from the browser for a resource that's under scope, we want this event to respond with a some code. Here’s our source, wiring of our fetch, then we do a respond with. Again this is something fetch this particular fetch event is going to respond with the following code.

First thing we want to do is try to find the request in the local cache. So we’re going to look for a match in the cache, see if there’s a cache entry that matches the event request.

Then at one point we’re is going to do, one of two things will happen at this point. At this point either the resource is in cache or it’s not.

So if the resource is not, sorry, if the resource is in the cache, we’re user going to simply return that response, the cache response back.

If it’s not in the cache though, then what we want to do, we want to go to the network and actually get from the network because we don’t have it available locally in the cache. So this is the cache first approach where we’re actually preferring our cache over the network, but we do go to the network to get the resource if we need to.

So when that request comes back, so we actually do go over to the network and make that request for that resource, when it comes back we want to store back in the cache. So we open the v1 cache to put that response into.

Then we’re going to go ahead and put the request, sorry, put the response into the cache using the request as the key.

So at this point our response is cached for subsequent requests, so now we can serve this same request again when it comes in this the future out of our local cache

So of course now we want to return the response, so we actually want to, so this particular event gets its response back. So we want to be sure this particular event gets an appropriate response, so we return that.
It's also possible that all of this fails. So for example if the resource was not in cache, and we don't have a network connection, then there's nothing we can do. So what we need to do in that scenario, or in this code snippet what we do, we just return a default image.

So this is an image that had been precached earlier during our install event.

This gives you just a really super high level overview of how service workers work. But there are certainly more nuances to this, but this should give you a basic idea of how it all works.

Cool. So service workers are great for caching URL addressable resources. So things that we can access over a URL, our html, our css, JavaScript, images, other assets like that. But if you're building an app, you probably also need a database to store things that are not URL addressable.

So for example let's say you're building a shopping cart app. You also need a place to store the items that a user adds to their shopping cart. That's where index DB would be really helpful. Indexed DB is basically a key value store that allows you to store data locally within a browser.

Indexed DB enjoys fairly wide spread adoption across browsers, so it's fairly safe to use, but there are a few things to consider about indexed DB.

It's not available quite yet in every browser. Even the browsers that do implement indexed DB have so inconsistencies in their APIs.

Fortunately there's a library called local forage that you can use that wraps indexed DB and provides a unified API that irons out those differences between browser implementations.

If the browser doesn't support indexed DB, local forage can actually wrap web SQL or local storage as an alternative.

This allows you to still take an off line first approach with your data, regardless of which underlying storage mechanism is supported by the end user's browser.

The API for local forage is pretty straightforward am it's a key value store. You can put them in and out using the same key.

It's a great way to store off line. It's a go to tool for off line data storage in a browser.

But there is one sort of challenge with local forage, and that's how do you get data into and out of the device.

And how do you move data in and out. Some people encounter this problem and go, well, easy, I'll just write in a sync engine. And if you have tried this before, some of you who are listening to this may have tried writing your own sync engine before, I just want to say one does not simply write their own sync engine. It's actually much more of a challenge than you might expect.

Fortunately you actually don't need to do that because there's a really good sync engine that already exists. It's called PouchDB.

Apache couch DB, an open source document database. It's been around about a decade now. And one of the most interesting things about couch DB is that it offers peer to peer replication capabilities, so it
has the ability to move data without the need to set up a cluster ahead of time. So you can just replicate data between two arbitrary couch DB databases at any time.

The neat thing about couch DB, of course there’s couch DB itself which offers great replication capability, but also database called pouch DB which is open source database that syncs. Pouch DB can run in a browser and sink with couch DB or anything else that implements the couch DB application protocol.

Of course there’s also IBM cloudant, so cloudant is a fully managed database as a service based on Apache couch DB. So if you don’t want to stand up and operate your own server, you can get an IBM cloud instance. We actually have an lite plan of cloudant you can provision within IBM cloud. This will allow you to try cloudant for free.

So I like this explanation of couch DB from Nolan lawsen, and he says couch DB's superpower is sync. Sometimes I try to explain by saying couch DB isn’t a database, it’s a sync engine, a way of officially transferring data from one place to another while intelligently managing conflicts and revisions. It’s very similar to GIT. When I make that analogy the light bulb often goes off.

I think this way of explaining is a great way of explaining what the power of couch DB is. And what you get with couch DB and related technologies like pouch DB and IBM cloudant.

If you’re building a progressive web app, you can use pouch DB in the browser. So this is pouch DB embedded in a web browser. And pouch DB actually just uses indexed DB, actually has an adapter to wrap indexed DB. So similar to how local forage works where it’s wrapping the underlying browser storage mechanism.

Like local forage, pouch DB also has other adapters, so if your browsers doesn’t support indexed DB, it can choose an adapter that is supported by a particular browser in which pouch DB is running.

So this is great because it gives you data stored locally in the browser that you can use off line first so you can read and write your data locally, getting that really fast user experience, then you can sync your data in the background with couch DB or cloudant.

The pouch DB API is pretty straightforward. You can see this example here from the pouch DB website. So this is API actually pretty straightforward.

You can create databases using this DB.click method. For example, this is change which is really useful, one of the most powerful features of pouch DB.

One thing I should note about pouch DB and couch DB, they are document databases, so they store JSON documents, basically just JSON objects.

Like I said this is a changes feed, which I think is really useful. Allows you to be notified whenever there’s a change in your database. So this actually enables you to take a sort of reactive approach to your application and actually changing your user interface, triggering user interface changes when something changes in your database.

This can either be a change that’s happened locally within the database, or it can be a change that’s been replicated in from a remote database.
As you can see in this example, they have also at the very last line replicated the local database to a remote database. You could also replicate from a remote database, and there's also sync method as convenience to will allow you to set up bi directional replication.

So I think this is a fun example of what you can build with service worker and pouch DB called poke dex.organize. It’s progressive web ab from Nolan law son. You heard the quote a few minutes ago. Nolan also one of the main contributors to pouch DB.

Kind of a really interesting story behind this pekedex website.

Nolan originally built as a native android app. Unfortunately the rights holders from pokeman sent a take down notice and he had to pull the app.

Then later this idea of progressive web apps came along and he was able to rebuild his poke dex app as progressive web app. Interesting thing is the rights holders for pokeman don’t care if the intellectual property is used in a website. They do care if it’s used in an app, but not in a website.

So even though poke dex.org offers the same fun at as a web app, because it’s technically a website, Nolan continues to keep this up and allow people to continue to use it.

So I’m a big fan of progressive web apps, as you can probably tell. I think they are an exciting new development. And I encourage people to explore what you can build today and what's possible with a PWA approach. But sometimes there are reasons why you might still need a native app.

So for example you might need access to certain device APIs that aren’t feasible on the web platform today. Or you might need to provide a certain native app like field in your application that wouldn't be feasible quite yet with a PWA, so you might not be ready to quite jump on the progressive web apps band wagon. That’s fine because fortunately there are schools like cordovi or ionic that will allow you to build your app using web tooling.

You can just html, cms and JavaScript and with one code base build your app and target multiple platforms such as iOS, android or other platforms.

One thing I want to note specifically about ionic that's really interesting is that ionic actually has a build target for progressive web apps. So you can actually with one code base, if you're using ionic, build for iOS, for android, and also build a progressive web app and deploy a progressive web app at the same time using that single code base. So that's a pretty neat feature of ionic.

So you can actually still use pouch DB even if you're not building a web app. So this pouch DB is just JavaScript. Pouch DB can actually run in pretty much any JavaScript environment.

It runs well for hybrid mobile apps. Pouch includes a couple of SQL light adoptions and this allows you to get around the typical limitationings in a web browser if you’re building a hybrid mobile app.

I should note alternatively we have a library called cloudant sync available for iOS and android. We have a mobile app that allows cloudant sync to work in a mobile environment. This allows you to use as alternative to pouch DB within your hybrid app.

You can sync that data in cloudant sync or on the previous slide to your couch DB or cloudant database in the cloud.
If you have the budget and resources, it can still make sense to build separate native apps for each target platform. So even if you still have a progressive web app and you’re still taking that approach, there can still be benefits to building a native app using that particular platform’s tooling.

So for example, if you’re trying to create a user experience that is most tailored to the end user’s platform of choice. For example think about what it would take to provide a really great user experience on an apple watch.

So some scenarios still makes sense to build a native app using that native app platform's technology. What iOS, android, windows or other platforms.

You can still take this off line for the approach. The data within the native apps.

As I mentioned previously, we have a cloudant sync library which allows you to embed a database in your iOS or your android app. You can store your data locally, have a database you can read and write from, and that database in the background can sync with couch DB or cloudant.

Also mentioned previously off line first is relevant to cloudant database as well. There's tool called electron that allows you to build using html and css and JavaScript.

Since electrons apps are build using JavaScript, you can also use couch DB within the electron apps. You can use level DB adapter or SQL light storage adapter. Two options available for couch DB databases with an electron app.

This enables you to build an off line first desktop app with data stored locally syncing in the background for an electron app.

One of the challenges with this off line first approach is how do you enable multi user apps.

The current version of couch DB unfortunately doesn’t supper document permissions. So once you give a user access to a database, that user has access to every document within that database. Fortunately there's actually a node JS back end called a hoody that makes building off line first apps a lot easier.

So hood.ie provides multi user capabilities, including ability to register, handle application of user, those sorts of multi user sorts of features. The way the hood.ie works, when a user is created it actually creates a new database for that user and ensures only that user has permission to access their own database.

The way this architecture would work, it’s a little different than the architecture you saw previously, is you would still have your data in an Apache couch DB or IBM cloudant database or instance, but then you would have your client, whether it’s pouch DB or cloudant sync, actually sync to a hood.ie served endpoint instead of directly with couch or cloudant.

This enables sign in and make sure each user has access only to their own database.

We have it in our native app or whatever we’re building, shipping syncing through hoody and having it use the features.
We have an experiment called cloudant envoy. Similar to hoody, it would sit between the client. Whether it’s couch DB or cloudant sync. Instead of actually creating separate databases for every user, the way envoy works, it actually still has only one database in the background, but it provision the illusion to the client, so to the couch DB or cloudant sync client, as having separate databases for each user. So all stored on one database but still have that separation of document and permissions based on who the end user is.

This is an option as well but still experimental, but it’s open source. You can try it out and let us know how it works for you.

If you like what you have heard today and you find this off line first approach interesting, I hope you will get involved in the off line first community. There are a number of different ways that you can do that. Definitely encourage you to check out our slack team. If you go to off line first.org/chat, you can connect with the team and connect with people working on different challenges in the off line first space.

Definitely encourage you to also follow at off line camp on twitter. So this is, we have a lot of interesting things we share. We’ll be announcing our next event hopefully soon as well.

Next I want to mention a really valuable resource which is the off line medium application. Really a ton of great articles and resources available on this medium publication. So we actually have session recaps and other material from past off line camps. This is a great way to get caught up on some of the discussions and topics that have been covered at off line camps in the past.

The URL is medium.com/off line camp.

We have had a number of really great discussions at off line camp, talked about progressive web apps and service workers, of course couch DB and pouch DB are common topics. We also have a number of or have had a number of really interesting discussions around the user experience of off line first. So how do you introduce these off line first concepts to help end user understand what’s happening with their data, where is the data, is it on the device. How much you expose to the user, what are some design patterns, user experience design patterns around off line first.

So some really interesting articles there as well on that topic.

The number of people in the off line first community are also interested in decentralized web, so making a truly peer to peer and decentralized internet and web, some topics on both those topics. Using technologies like DAT or the experimental beaker browser, a number of other related technologies.

Also a lot of really interesting discussions around how off line first is relevant to developing world and how this approach can help in a number of different ways.

One example of this is an interesting open source project called hospital run. It’s an off line first app that’s used for healthcare in Africa. So what they used to do, they had a project where they would load up patient records on top of a Jeep and drive out to a village somewhere, but now they can actually store all that data in an off line first way on a device, take down medical records, get information from patients, then sync all that data back when they get back to headquarters.
Then final one of the things that I really love about off line camp is that we get a lot of really unique and interesting people with a wide variety of interests. One of the fun things we do at off line camp are what we call passion talks. This is where we ask participants to do quick five minute talks on any topic at all, something they are passionate about. Doesn't have to be about off line first. Most of these talks or a lot have been recorded, and you can find them on the off line camp medium publication as well.

Of course we have links to a number of other resources we have gathered there, a great jumping off place for learning what off line first is all about.

Great, so with that, at this point I want to open the floor to any questions folks might have. We actually have my colleague, Mike bromberg, here on the call as well. He will be moderator any questions that you might have for me.

>> MIKE: High everyone on the chat. Thanks for lively discussion. Bradley, we had a few questions I wasn't able to address off the top of my head. I'll start from the first one, which is we know that PWAs, you can grant them access to certain device specific resources such as your microphone and camera. I had a question come across chat if you could grant access to the fingerprint readers as well. Would you happen to know about that?

>> BRADLEY: Great question. I don't know the answer to that. I have not heard of any. I haven't heard of an API for that. But I don't know for sure that there isn't one. If there is, I haven't heard of it.

>> MIKE: Yeah, something for us to look into.

When you were registering the service worker and showing basically resources you were adding to the cache, I had a question come up, is there just an easy quick shorthand way to cache all the contents of an entire folder?

>> BRADLEY: Great question. So the short answer is I don't think, I think the question is is there a wildcard or some way to do that.

>> MIKE: I think that's the question, yeah.

>> BRADLEY: Yeah. I don't think there is. But there are tools. The name is escaping me at the moment, but there are definitely tools that make worker with service workers easier that you can build as part of your build process. Workbox I think is one of them and a few others. Different frameworks have tools too. I know polymer has tools for this. There's tooling you can build into your build work flow that will make that easier. So you can basically, you don't have to go write all those resources out in your service worker. It will build your service worker file for you automatically as part of your build process.

>> MIKE: Cool.

So there was another question on hood. Ie, and basically does hood.ie support a multi database model similar to envoy. Can you speak to how hood.ie is storing things in couch DB behind the scenes? I wasn't quite sure of the structure.

>> BRADLEY: Yeah, hoody and envoy take a different approach. Hoody takes the one database per user approach where it actually creates a different couch DB database, if you're using cloudant, cloudant database, for every single user. If you're using other databases, that might seem strange at first but works surprisingly well and scales to fairly large numbers.
There are limitations like if you want to do aggregation across users and stuff like that. Definitely some challenges in there to still be figured out. But that's the approach hoody takes where you're actually segmenting every user into their separate database.

Whereas with envoy, envoy provides that same illusion. So a client replicating through envoy or through hood .ie wouldn't know the difference. With envoy it would think it has its own database, replicating with a database only that user has access to, but in fact envoy is actually keeping all that data, all those documents in one single database in the background.

So you don't actually have those, you know, separate databases for every single user. But while you're still maintaining separate missions and pools of data, if you will, from an end user perspective.

>> MIKE: Cool. This one, thank you, by the way. This is more just a comment from the chat, which was someone mentioned that they were a huge fan of cordova and ionic but was a bit of bummer when cordova pulled support for fire OS.

That was something I wasn't aware of. I guess kindle fire OS.

Just wanted to mention that.

Shoot, one other point that I wanted to make.

Oh, yeah.

I think the logo for Google's light House flashed somewhere in your presentation. Or maybe it didn't and I'm making that up. But the notion of kind of testing or test automation for PWA came up a couple of times in the chat. I wasn't sure if that's something you wanted to kind of comment on here and we can kind of close it out after that.

>> BRADLEY: Yeah, probably should have mentioned that. Great point. Lighthouse is a great tool. Basically what it does, it lets you test your progressive web app against a number of different benchmarks, gives a score on a number of dimensions. Things like does it work off line, have a service worker, accessibility score, a number of different dimensions.

So it's actually a very, I would definitely encourage you to use lighthouse. You can get it as a chrome extension. Also a way to automate and build it into your build process as well at the command line. But yeah, a really great point. Lighthouse is an excellent tool. If you are building progressive, you should definitely familiarize yourself with lighthouse.

>> MIKE: Thanks, thanks everything unresolved from the chat.

>> BRADLEY: Great.

>> Thank you, Bradley and Mike for that presentation.

One question for you, bradly, if you don't mind. Can you hear?

>> BRADLEY: Go for it, yes.

>> As far as the pattern is concerned, for somebody to really experience the flavor of off line first, how easy is it for them to run that pattern and start playing with it? Is that a good way to start?
BRADLEY: Yeah, those shopping limitations, you can get those off Git hub and actually run them yourself. We have instructions for getting it up and running for each of them. We also have links to live demos at least for progressive web app ones, we have them on Git hub pages. If you don’t want to run in your own environment. All the source code is there.

If you’re react, check that out. If our polymer, check that out. We did it in different frameworks so you can click around the one that’s closest match for your tech stack of choice.

>> I think that would be a great way to start. That’s a very important technology, the ability to do that. As you were relating, in developing companies, that’s a life saver to be able to do that. You can also find use cases in developing countries. I think that’s a great thing.

Do you know where the next camp is going to be?

>> BRADLEY: We don’t. Hopefully announcing soonish. We actually haven’t picked a location yet. So once we have a location settled, we’ll let people know. Definitely keep an eye and follow at off line camp on twitter and we’ll let folks know.

If you want us to go to somewhere in particular, you can go to off line first.org/camp, there’s a link where you can actually fill out a form and let us know where you’d like to see us take it next.

>> Interesting. I’d like to participate.

Can you give us a quick rundown of what off line camp is about? Do you take beginners? How does the camp run?

>> BRADLEY: Off line camp, we have a whole range of people, everyone from people fairly new to off line first, really just interested in it, want to know what it’s all about, to people that are building the technologies like the contributors to pouch DB, couch DB contributors, people actually building these tech stacks we’re working with. So it’s an interesting range.

Like I said, the entire event is run by the participants. So we actually have people propose sessions and vote on those sessions, and then we have group discussions on the various topics that have been selected. So definitely encourage people to check it out. Would love to see folks at our next off line camp. For me it’s been a really great event to be a part of. A lot of fun, informative, super interesting discussions and people there.

>> Excellent, very good.

I think you sparked a lot of interest today with this technology, and I hope people will sign up for the next one and become proficient and start doing off line first. Certainly a great thing.

>> BRADLEY: Thank you everyone for joining. Appreciate the interest.

>> Thank you, Bradley. Tomorrow we have another tech talk at 1 p.m. eastern, and it’s about decentralized energy with hyper ledger composer. A lot of interesting people using solar panels to sell excess power to the company. That’s going to be a great presentation also tomorrow. So join us tomorrow at 1 p.m. eastern. We will have Rajo berry from the block chain team who will take us through this. That’s going to be an interesting one.

Thank you for your time, and we’ll see you tomorrow.
Bye bye.

(Session ended at 12:53 central).