https://ctrlclickcast.com/episodes/offline-web-experiences

CTRL+CLICK CAST #121

Offline Web Experiences with Jeremy Keith

CTRL+CLICK CAST is proud to provide transcripts for our audience members who prefer text-based content. However, our episodes are designed for an audio experience, which includes emotion and emphasis that don't always translate to our transcripts. Additionally, our transcripts are generated by human transcribers and may contain errors. If you require clarification, please listen to the audio.

Preview: And the web still feels, I think in most people's minds, like it's dependent on a network, and so service focus introduced us kind of where to I think where it's like, "Yeah, I know the network is there, we're totally going to use it, but we're not totally dependent on it now," and it could be getting closer and closer even to what you came to expect from native where you're in a situation you know don't have a good internet connection, but you're still going to open up that website and expect to be able to like, I said, read something. And being able to read some text, that does not sound like a big request here, that sounds like something we should be able to handle.

[Music]

Lea Alcantara: From Bright Umbrella, this is CTRL+CLICK CAST! We inspect the web for you! Today Jeremy Keith joins the show to discuss offline web experiences and progressive web apps. I'm your host, Lea Alcantara, and I'm joined by my fab co-host:

Emily Lewis: Emily Lewis!

Lea Alcantara: This episode is sponsored by toku — the digital marketing and development studio of Luke Stevens, launching October 1st. Luke loves Craft and has been using ExpressionEngine since 2004. Does your agency need help with a CMS build, front-end work or expert SEO analysis? Email luke@itstoku.com or visit itstoku.com for more.

[Music ends]

Emily Lewis: Before we get to today's topic, I wanted to remind our listeners we have a donate link on our site. So if you love CTRL+CLICK and have a little spending money, consider donating to help us keep the show going, a dollar, five dollars, whatever you can spare will help continue to deliver great content, high quality audio and transcripts for each and every episode.

Now to our guest, and I have to admit, I'm having a little bit of a fan girl moment. [Laughs]

Lea Alcantara: [Laughs]

Emily Lewis: Since the beginning of my exploration into the web and teaching myself how to code, I've looked to Jeremy Keith and his blog and his books. I've always admired his commitment to learning and teaching as well as his practical user-focused perspective. So to say I'm excited to talk with him today is an understatement. Jeremy lives in Brighton, England where he makes websites with Clearleft. You may know him from such books at *DOM Scripting*, *Bulletproof Ajax*, *HTML5 For Web Designers*, *Resilient Web Design*, and most recently, *Going Offline*, which is what we're going to talk about today. I'm so happy to have you on the show, Jeremy. Welcome.

Jeremy Keith: Thank you very much for the lovely introduction.

Emily Lewis: [Laughs]

Lea Alcantara: So Jeremy, can you tell us a little bit more about yourself?

Jeremy Keith: Sure. I live in Brighton, England, though I'm originally from Ireland.

Lea Alcantara: [Agrees]

Jeremy Keith: I've been living in Brighton for just about 18 years now. Yeah, like you said, I'm working at Clearleft agency, it's kind of nice, down by the seaside, and in my spare time, I like to play Irish traditional music. I'm usually playing on the mandolin or the bouzouki.

Emily Lewis: And how did you even get started working on the web? How did your education or career take you in that direction?

Jeremy Keith: Well, it started in the 90's. I was selling bread in a bakery in Germany, and so the obvious next step was making website.

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: As well as selling bread in a bakery. I was playing in a band, and the web was starting to be on everyone's radar and we decided we should have a website, but nobody knew you made a website.

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: And I thought I'd give it a go, and I did a little research and asked around and kind of fell in love with the power that comes with typing characters into a text file and then saving it and then seeing it inside a web browser and publishing that for the whole world to see. Yeah, and so I started with making a website for the band, and then other people started asking if I'd make websites for their bands and even offering to pay.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: And I realized you could make money from this. I was freelancing then for quite a while and then moved to England and was freelancing here as well, and yes, I set up with Clearleft in 2005, but yeah, basically, music was the conduit to the web.

Emily Lewis: And I kind of mentioned in the intro, you've been writing and teaching through your writing for as long as I can remember. Well, how did that get started? What got you into publishing books and writing on your blog in a way that's, you know. You have sometimes a very philosophical way of looking at things, but generally speaking, you're really practical and so the things you share are things that people can put to use right away.

Jeremy Keith: Well, when I was getting into the web, yes, in late 90's, it was sort of before blogs and everybody writing stuff, but my website back then was more like a portfolio site, and it was around 2001, I kind of switched over to making it more bloggy, and I have this thing remembering at the time thinking, "I'm so late to this."

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Like as a follow up, blogs had already happened. This was 2001, and I felt like, "Oh, I'm so late to this bandwagon, this whole writing blog post thing."

Lea Alcantara: Ah…

Jeremy Keith: But of course, it's never too late. When I still meet people today, they say the same thing, "Oh, it feels like there's no point starting now," and I'm like, "Listen, I thought that back in 2001. You can totally start now." But partly it's because when I was learning how to make website, I was finding out by reading what other people have written and people were just very open and sharing by default, it felt, and so it just felt natural as I started to learn stuff and realized, "Oh, I've got an opinion here. I've got a technique I could share," whatever it happen to be, the idea that "Oh, I should share this publicly" felt very natural. It felt like the right thing to do on the web. It didn't feel like a weird thing to do, and so I kind of just fell into it.

And as I started to write my own website and did that more and more, it kind of gets addictive. It's kind of just fun to do. It just feels good to be writing about stuff. It's not always about web stuff. I try write about anything I feel like really, but yeah, it just felt like the natural thing to do, and then the books came much later. My first book was in 2005, but I already had a blog for years at this point, and with the books, they generally come out of frustration where it was like, "Somebody should write a book about this."

Lea Alcantara: [Laughs]

Jeremy Keith: And you realize, "Oh, wait, I'm somebody, and if nobody else is going to do it, I better do it, right?" So that feeling of like, "I'm excited about this thing and why isn't anybody talking about this," and then realizing I could be the one talking about this.

Lea Alcantara: [Agrees]

Jeremy Keith: So I think that had been the genesis of when things get put into book form.

Emily Lewis: [Agrees]

Jeremy Keith: Now, a little over time, you would have noticed, if you look at my first book and then the book after that and the book after that, they get shorter and shorter each time.

Emily Lewis: [Laughs]

Jeremy Keith: Like I think they have every time in length.

Emily Lewis: Well, we're going to dive into talking about your book in a second, but just to that point, I'm actually looking at *DOM Scripting*, and you're right, it's like a textbook, but your *Going Offline*, which is from A Book Apart, I actually got a digital book. I got through that very quickly.

Lea Alcantara: [Agrees]

Emily Lewis: I'd say maybe four hours of time and then going back and sort of testing some of the scripts after I read through it, and so it was nice to be able to absorb something that, frankly, I was like, "Ugh, this is going to be a hard one for me to understand," but it didn't feel that way at all and to be able to get through it so quickly was nice. So I kind of like this direction of these really consumable books, like you can just sit and read it, maybe with a cup of coffee or something, and then go back to it later to try to put it to work.

Jeremy Keith: Yeah, I think the length of the A Book Apart series is definitely a feature, not a book.

Lea Alcantara: [Agrees]

Jeremy Keith: And I know I've got a short attention span, so I can't remember the last time I cracked open like a proper big computer book.

Emily Lewis: [Agrees]

Jeremy Keith: I don't think I could take it anymore.

Emily Lewis: [Laughs]

Jeremy Keith: I kind of need things to be in short adjustable formats.

Lea Alcantara: So do you think that there's no more room for the length of something like *DOM Scripting*?

Jeremy Keith: I think maybe that's the right format for — what would you call like reference books, right?

Emily Lewis: Yes.

Lea Alcantara: [Agrees]

Jeremy Keith: Where it's like an A to Z of everything and you look stuff up.

Lea Alcantara: Right.

Jeremy Keith: I'm not so sure about reading something cover to cover for computer stuff basically.

Emily Lewis: [Agrees]

Jeremy Keith: I mean, I obviously read fiction. I read nonfiction that's plenty long, but when it comes to I guess work-related stuff, I would kind of expect the bigger books to be reference books.

Emily Lewis: Yeah, I actually opened up my own book last week for reference. This was like, "Oh, I know it's in Chapter 5 so I'm just going to look up there really quick."

Lea Alcantara: [Laughs]

Jeremy Keith: Oh, yeah.

Emily Lewis: I don't know what's even going on.

Jeremy Keith: I've done it so often, it was like, "I can't remember this JavaScript thing, but I know I wrote about it two years ago. I've got a book on it."

Emily Lewis: Yeah. [Laughs]

Lea Alcantara: Can I just interject a little bit, too? I think what both of you guys are saying is so important to kind of emphasize, especially when people go to these like coding tests for job interviews.

Emily Lewis: I wouldn't pass. [Laughs]

Lea Alcantara: Well, you know what I'm saying, right?

Emily Lewis: Yeah.

Lea Alcantara: Like as in like, do you need to memorize every single thing? The main thing is like understanding how to solve the problem in however way possible, not memorizing every single solution.

Jeremy Keith: Absolutely, yeah. Whether it's CSS or JavaScript, or even HTML, knowing what's possible is important, but knowing the specific like what's the literal name of that thing, you don't need to keep them in your head, right?

Emily Lewis: Yeah.

Lea Alcantara: Right.

Jeremy Keith: That's what Google is for.

Emily Lewis: Yeah, exactly.

Lea Alcantara: Yeah.

Emily Lewis: So let's get into your latest book and start with the basics. So what does offline mean when it comes to websites, Jeremy?

Jeremy Keith: Well, that's a good question because I guess the thing that immediately springs to mind is when the user, the person trying to access a website doesn't have an internet connection so they're the ones who are offline, and that's more and more common as we switch to mobile and not even just in developing countries and stuff, but in just about anywhere involving public transport, right?

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: Inevitably, there's going to be a time when you as the user are offline, but there's another side to offline, too. It's less common, but it's worth considering, which is that sometimes a website goes offline as in the server is down for some reason, and to the end user, there isn't much difference between "Oh, I've lost my internet connection" or the web server for this website is down.

Timestamp: 00:10:00

Lea Alcantara: [Agrees]

Jeremy Keith: In both cases, it's offline.

Emily Lewis: And so in those situations, I think a lot of us, just users, not necessarily the developers, we kind of are like, "Okay, well then, I'm just not going to utilize that resource right now." But it's becoming more important that we do need to let users access that information when they're still offline. Why is it having a good online experience where you users can still get info rather than just some sort of error message?

Jeremy Keith: Well, there's the immediate benefit, which is that, "Oh, I can still accomplish what I wanted to do even though I don't have an internet connection or even though the web server is down." So that's the immediate individual benefit, and that's important. But you touched on something there, which I think speaks to the bigger question, which is expectations around what you expect from the web, what you expect when you open up a web browser, and you're right, right now, I think the expectation is, "Oh, I'm offline or my internet connection is down, so I won't be able to do anything. Because that's how the web works, I can't do something if I'm offline."

Emily Lewis: [Agrees]

Jeremy Keith: But if enough of us do build experiences that work offline as well as that individual benefit of okay, you get to accomplish what you want to do, there's this added network effect of we're helping the perception of the web get to a place where people do thing, "You know, I'm offline, but I'm still going to try to accomplish what I wanted to do."

Emily Lewis: [Agrees]

Jeremy Keith: That the expectation of what's possible in the web goes up to include being able to do stuff when you're offline.

Emily Lewis: And you mentioned a little bit about rural connectivity. I think something else came up recently from Eric Meyer. He was writing about his experience. I believe he was in Uganda or something like that. He was trying to load a page and nothing came up and there was an issue because of HTTPS. It prevented the site from caching and loading. I mean, nothing was loading based on his experience, and so we, Lea and I, with our clients, have been really getting a lot of SSL certificates installed on site and stuff, and it's always been from a very positive perspective, but Eric's post kind of brought up a new concern that HTTPS might contribute to this problem of accessing information when the network is not good.

Jeremy Keith: Yeah, it was a fascinating post by Eric and very evenhanded. It wasn't in any way anti-HTTPS.

Emily Lewis: [Agrees]

Jeremy Keith: He was just relaying what he saw, and there was a great follow-on post by Tim Kadlec who kind of dived deeper into this. So the issue was in these rural areas where satellite connections, in order to save on bandwidth, which is extremely thin on the ground and pricey…

Emily Lewis: Expensive.

Jeremy Keith: They would have caching servers, so once you've accessed something once, they could store a copy so that the next time you'd try to get to that, it didn't have to go all the way up to the satellite, it could just go through caching server. Now, technically, that's spoofing the website, right?

Emily Lewis: [Agrees]

Jeremy Keith: So HTTPS sort of cuts out that possibility.

Emily Lewis: [Agrees]

Jeremy Keith: It's technically a man-in-the-middle attack, and so now, they can't use the caching servers and so now they're having to go to the satellite every time, but what Tim really dived into was it wasn't that HTTPS was the problem, it was that we've added security, but we didn't fix the fundamental issue which is around performance.

Emily Lewis: [Agrees]

Jeremy Keith: So why were they caching these pages in the first place? Why did they use a caching server? Oh, because these pages are too big to be downloading fresh every time, and so caching server is a good solution there. So the problem being addressed was performance and what gotten in the way of that was security. Now, security and performance don't need to be at odds. In fact, they really, really shouldn't be, and Tim's point was kind of that having security without performance is going to hurt people, and that's exactly what Eric was describing.

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: What we really need is a balance; security, performance, and accessibility is the other sort of pillar I think of universal access on the web. So what was happening here was kind of a bit of a mismatch. There was an over-emphasis maybe on security or not much to do with over-emphasis, but there wasn't an equal amount of emphasis on performance as there was on security.

Emily Lewis: [Agrees]

Jeremy Keith: And he's absolutely right that if you're going to tackle something first, I think get your performance in order first and then start thinking about things like offline access and security and stuff like that.

Lea Alcantara: [Agrees]

Emily Lewis: That's a good point. So let's talk about offline access. I guess I wanted to like have some sort of reveal here, but really it's about service workers, and that is kind of the large focus of your book, how to write with service worker, what function it provides to give that sort of offline access. So can you describe what a service worker is and how it supports a good offline experience?

Jeremy Keith: I can try, but I'll tell you, they're weird things to get your head around.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Actually, that description of Eric's experience in Africa might be a good reference point, but they were talking about having caching servers so there's a server that's keeping copies of websites, effectively web pages, and you're hitting that instead of going all the way to the network.

Emily Lewis: [Agrees]

Jeremy Keith: Well, a service worker is kind of like a caching server. It's not anything a server or an actual machine somewhere, it's inside your machine, it's in your browser, and so the same way I described the caching server is kind of being like a spoof, a man-in-the-middle attack, that's sort of what a service worker is, except it's a good kind of spoof.

Emily Lewis: [Laughs]

Jeremy Keith: Because you're doing it to yourself or you can only do it to your own website. You can only kind of hijack requests for your own website. So it's this kind of caching proxy that's sitting on the user's machine, and so when they request a URL from your domain, instead of it immediately going out to the network to fetch that resource, it checks first with the service worker, and the service worker is written in JavaScript. It's a script. Now, that can be confusing as well because people are like, "Oh, right, JavaScript files. I know how to treat JavaScript files. I'll add it. I'll concatenate all my JavaScript together and putting to one big folder."

No, not with service workers, it needs to be separate. It needs to be its own file. I almost think like it happens to be written in JavaScript, but you don't want to treat it like all your other JavaScript files. It's a bit different. It's a set of instructions that happen to be written in JavaScript, and so the end user's browser looks up the set of instructions first, and a set of instructions might say, "Yeah, go to the network, that's fine, do that." But a set of instructions might say, "Well, before you hit the network, try this instead, and that might be try looking in this cache I created earlier received as a copy of the resource there or yes, serve the version from the cache, but then immediately go out to the network

afterwards to get a fresh copy." It depends on what you're trying to balance there with freshness, speed of response, all this stuff. So basically, it is a set of instructions that sits on the user's machine and gets consulted before any requests go out to the network.

Emily Lewis: And so when someone is in an offline experience, if there is a service worker for that site, then depending what those instructions are, but if those instructions are, "Hey, serve a cached version on this page or this resource," they're going to get it and it would be seamless to them. There's no like, "Hey, we're loading a service worker alert." It just happens.

Jeremy Keith: Yeah, and that's key as well that a service worker doesn't give you anything for free. It's not like, "Oh, I have stalled a service worker, now I'm going to get an offline experience." It's like, no, you have to write those instructions, just like you said. So the instruction might be, "Yeah, try and go to the network every time and only if that fails, which would hint that there's a connection problem here, then here's a fallback solution which might be to show a cached version of that page if we have it or show a message saying you're offline, but here's a game you can play or something like that." Yeah, you have to write instructions. With the service worker, you don't get anything for free. You have to think about what you want their experience to be and then translate it into JavaScript instructions.

Emily Lewis: So I think one of my favorite analogies in your book was how you were initially describing service workers, that it's kind of like a cookie, it's kind of like a virus, or it's kind of just like a toolbox. So for me, I really connected with the cookie aspect because I understand that as a user that those kind of reside on my phone or my computer because I visited a site and it basically installed it in the background, and in a lot of ways, that is how a service worker works. When you first visit a site, if that site has a service worker, it kind of gets downloaded like a cookie, right?

Jeremy Keith: Yeah, exactly, and that's so hard to explain. I was trying to reach for things that might be familiar to help explain them, and none of them really work entirely because it's only sort of

like a cookie, but it's obviously way more powerful than a cookie, because you write these instructions, and it's only sort of like a virus because it's benevolent, you're the one writing the instructions, but maybe by putting together a few different analogies like that, you can sort of piece together, you start to get your head around the idea of what a service worker is because it does feel unique in the sense of I've been making websites for decades and this idea of this file that gets installed in this machine and it's executing the JavaScript, it took me a long time to get my head around that.

Lea Alcantara: Well, do you think that is it because the nature of the web, we're always thinking online, and like earlier n this conversation we were talking about rural connectivity, but offline experiences are actually happening more and more, as you said, in transportation. For example, the first thing I think about is, I'm sure you travel a lot to speak at conferences, plane WiFi is the worst. [Laughs]

Jeremy Keith: [Laughs]

Lea Alcantara: And to me, I just think that the majority of the time, if I'm not doing something else for work, I'm surfing the web and all I need to do is read something and if it's not properly cached or if there was no specific thought to potential offline experience, then it basically makes airline WiFi completely useless.

Timestamp: 00:20:08

Jeremy Keith: Yeah, I think it has to do with that expectation, right?

Lea Alcantara: Right.

Jeremy Keith: When we think of the web, when we think of being online because that's the web is about and we think of situations where you're not online or you don't have that connection, you just assume, "Well, the web won't be able to cope with that. That's not what the web is good at."

Lea Alcantara: [Agrees]

Jeremy Keith: So there's this whole level of expectation setting that I frankly think is the really hard work here. Writing service worker script I think isn't that tricky. Thinking about what you want to experience to be I think is quite manageable, but altering the general expectation of what people have come to expect of the web, that's huge. That's a big, big task, but hopefully, we can. Again, with making comparisons and analogies, people don't tend to, I have to say, an expectation when it comes to native apps because of this idea that you installed something and so it's on your phone, right?

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: So you do think, "I should be able to open it and be able to do something even if I don't have an internet connection." Right?

Emily Lewis: Right.

Lea Alcantara: Right

Jeremy Keith: So this idea of installing feels like it comes with an expectation of independence from the network, right?

Emily Lewis: [Agrees]

Jeremy Keith: And the web still feels, I think in most people's minds, like it's dependent on a network, and so service workers introduced this kind of weird two-way thing where it's like, "Yeah, I know if the network is there, we're totally going to use it, but we're not totally dependent on it now." And it could be getting closer and closer even to what you came to expect from native where you're in a situation you know you don't have a good internet connection, but you're still going to open up that website and expect to be able to, like I said, read something, and being able to read some texts, that doesn't like a big request here. That sounds like something we should be able to handle.

Emily Lewis: So let's talk a little bit about the technical aspects of a service worker. So it's written in JavaScript, but it's treated differently. Can you explain how it's different from maybe the JavaScript everyone is used to writing? For one, I know that JavaScript is so far outside my comfort zone, but if it's written in the latest version of the language, then that's okay because essentially this is a service workers' progressive enhancement. You're really only adding it on top of your foundational stuff and only browsers and devices that can recognize it are going to understand a service worker, and same with the latest version of the JavaScript language.

Jeremy Keith: Right. So it's this kind of a nice happy coincidence that the service workers are relatively recent technology and the new additions to the JavaScript language, in the ES or whatever we're calling it now; ES2015, ES5 or whatever, E6, they're also relatively recent, so there's this nice Venn diagram crossover, which is if a browser supports a service worker, it supports these new ways of writing JavaScript. It's because when you tell the browser to install the service worker, you first kind of test. You'd kind of say, "Wait, do you know what a service worker is? If a service worker exists, then install the service worker."

Emily Lewis: [Agrees]

Jeremy Keith: And only then will the file be called. So you know for the browser to get past that test, we're dealing with a modern browser here, and so you no longer have to worry about having fallbacks

for older ways of writing JavaScript. So that's kind of nice if you are going to dip your toes into these newfangled JavaScript things. I mean, they're not that too new anymore. They're been around for a few years, but still, you know. If you've been weary of trying some of them out because you quite rightly are thinking about backwards compatibility, a service worker is kind of a safe space to play around with this stuff, so that's one thing. But then with all your regular JavaScript files, you might do something like have a folder with separate files and then have a build tool that takes them and mushes them all together into one file.

Lea Alcantara: [Agrees]

Jeremy Keith: You wouldn't want to put your service worker script in with all those files. It kind of needs to be separate. It needs to be its own script. And also, let's say you put your usual JavaScript files into a folder, maybe you have a folder for CSS, it's like /css or /assets/css. And for JavaScript, it's the same, it's maybe /js or /assets/js, and you think, "Right. Well, service worker file, that's written in JavaScript. I'll put it in my /js/ folder." Well, you don't want to do that because by default, the scope of the service worker, in other words, which URLs it's looking out for will match where it lives, where it sits. So if you put it in the /js/ folder, it will only be able to handle requests that go /js/ something.

Emily Lewis: [Agrees]

Jeremy Keith: So that's what you generally want to put it in the root of your sites if you want it to be able to handle any request for anything on your website. So that's one more reason why even though it's written in JavaScript, you don't want to treat it the same way you treat your other JavaScript files; hence, that's why I think it happens to be written in JavaScript, right?

Emily Lewis: [Agrees] And in terms of being able to use service worker in the first place, what are the requirements for the environment to support that? I mean, I'm not referring to like browser support, but for you to serve a service worker. Are there some requirements on the server end?

Jeremy Keith: Just that if someone visits your website, that's pretty much it.

Emily Lewis: Doesn't it need to run on HTTPS?

Jeremy Keith: Oh, you're right, of course. Yes, yes, of course. I've gotten so used to it now, I don't even think about it, but yes, you can't use a service worker on an HTTP regular connection, which you know thinking back to what we're saying about it being a man-in-the-middle attack on your own website, that makes total sense because if someone could spoof your website, they could install a service worker and that would be really, really bad. So you're right, of course, yes, HTTPS is the requirement, which kind of going back to Eric's point about the unintended downside of HTTPS being that, meaning now we can't use a caching server, but as he pointed out in his post, service workers are kind of an option and alternative to that site. Well, the thing as you switch to HTTPS, now service workers have opened up to you and you could replace the functionality you are doing in a caching server and do it in a service worker instead. So I mean, that will be nice if the move to HTTPS was accompanied by a move to having some kind of service worker as well.

Emily Lewis: So for me, I tend to struggle with JavaScript that I'm not familiar with when it's brand new, and I thought it was pretty easy to take your examples and apply them, and we're actually working on a mobile redesign of CTRL+CLICK's website, so I'm going to just go ahead and put a service worker into that whole launch, and so I thought it was really easy to follow along and build it. I felt like the way you chunked it was really useful, kind of starting really small and then building from there. But I did think it was, you know. It took me a beat or two to get used to how you update. It's not the same as like clearing my browser cache, saving a file, making changes, saving the file,

clearing my browser cache. It's a little bit different from typical front-end workflows. Can you describe that a little bit?

Jeremy Keith: Yeah. This is always an issue with anything to do with caching when you want to sort of flush the cache. Now, whether this is something you're doing on a server or was doing in a service worker, anything around caching always is false positives and trying to clear a stale caches or is that's an old saying that the two hard problems in computer science are caching validation and naming things, right?

Emily Lewis: [Laughs]

Jeremy Keith: And that certainly, service worker is no different. I will say it's better than what we had before. There was a previous technology around trying to make offline experiences called AppCache, which if you ever try to use it, it was a nightmare, and you could very easily end up with a permanent unkillable cache where there was literally no way for your users to ever see fresh content again. It was nightmarish.

So with service workers, you do have to consider like, okay, when you're developing, making sure, "Am I looking at the latest version of the service worker in this browser now?" And browsers have some good tools from that. Chrome especially has a nice panel under Application in Dev Tools where you can kill the current service worker, update it right there and do stuff like that.

There's also some sort of failsafe that's built in. So let's say you set up a caching strategy. Whether it's in your service worker or even on your server where you say, "Hey, I want JavaScript files to be cached for years," and now you update your service worker. Now, wait a minute, service worker is a JavaScript file and you've told this browser to cache JavaScript files for years. There's a failsafe built in, which it used to be, I think it was 24 hours I think the service worker, it could only be cached for a maximum of 24 hours before the browser would check again for a fresh version, so it completely

ignore that instruction that said, “Cache JavaScript files for years.” It would basically treat it as an exception. I think that 24 hours has even gone down now to just every time it’s decided it’s going to check to see if there’s a new version of the service worker.

So there are things in place to stop things getting too out of hand there, but when it comes to developing and debugging, it can be a bit of a pain and you’re constantly having to make sure you’re killing the instances of the current worker. Like if you have two tabs open or two browser windows open with your website, it’s not enough to just close one of them, you have to close all of them in order for the service worker to kind of go away. So yeah, there are lots of gotchas around the life span, I guess, with the service worker and in validating the current version and updating it, but there’s also this failsafe built in to make it easier based on the lessons we learned from things like AppCache, which definitely were anti-patterns. We knew like this is how we know we don’t want to do it, so let’s do it differently.

Emily Lewis: So you sort of touched on it a little bit, like referring to someone saying, “Save the JavaScript files for like years and years.” So where does a service worker in the caching that you can do with the service worker fit in with HTTP caching? Or even this is still server-side caching essentially, but we would use ExpressionEngine, for example, to cache pages that it’s generating in kind of like save a static file on the server or whatever. Can those coexist with service worker or should only one exist?

Timestamp: 00:30:17

Jeremy Keith: No, they’re definitely complementary, and I would think whatever you’re doing before you came along and added the service worker, you should probably keep doing. Having a service worker doesn’t negate the benefits of server-side caching, HTTP cache, and all of that stuff. The service worker is kind of like a layer on top. In fact, probably a good way of thinking about service worker in general, it is just an enhancement. It is something that you layer on top of what you are

already doing and you shouldn't have to go back and re-architect your website or change your caching strategy, it should sit on top of that.

Emily Lewis: [Agrees]

Jeremy Keith: Now, what's worth remembering though when let's say you write the instructions to your service worker and you might say something like, "Okay, check and see if this page is cached in this cache we created earlier. And if it isn't, go out to the network." Well, technically, when you say go out to the network, it's not so much that it literally goes out to the network, it's more like you're saying, "You know, look into this cache, or do what you would have done anyway." Now, if what you would have done anyway was look into HTTP cache, then that's what will happen. So that's worth bearing in mind as well that when you say, "Use the Fetch API and say fetch this from the network," what you're really saying is do what you would have done anyway, which might mean you're still getting a cached version, but it's cached on the server or it sits from the HTTP cache so that's worth bearing in mind.

Emily Lewis: [Agrees]

Jeremy Keith: So if you have some strategies in place, for example, with CSS and JavaScript files, you might be sending instructions from the server and have it to say, "Yeah, cache these files for months or years," and then when you update your CSS and JavaScript, you've got some kind of cache-busting strategy like you change the file name, you add in a query string, whatever it happens to be, you would still need to do that with a service worker in the mix.

Emily Lewis: [Agrees]

Jeremy Keith: So you could add in this other layer of caching with service worker, which is like, "Oh, yeah, let's cache the JavaScript and CSS so that we can serve them up without ever going out to the

network," but the same rules still apply, you should still be doing it with some kind of versioning in there, some kind of query string or version number, and have something in place in the service worker as well to check on like, "Is this the latest version? If not, go fetch the fresh version."

Emily Lewis: So when you're creating a service worker, what do you think the minimum experience someone should be aiming for?

Jeremy Keith: So that's a good question. I was thinking about it like, "Is there a minimum viable service worker, like a no brainer stuff should be good?"

Emily Lewis: [Agrees]

Jeremy Keith: And I think there are some performance benefits that pretty much any website could benefit from. Like we were just talking about caching your CSS and your JavaScript, and maybe icons as well, I mean this is all the stuff you're trying to do anyway with the server cache, with like server headers, so yeah, do that for sure as a bare minimum, precache CSS, JavaScript, icons, stuff like that. Beyond that, that's where it gets interesting.

I think there's another sort of no brainer you could add in, which is this idea of an offline page. So usually when the user is offline or the website is offline, it's the browser that displays a message. It's the dinosaur game in Chrome or what have you, and with service workers, you can have a page precached, that's your custom offline page. Now, in some ways, that's sort of like having a custom 404 page. It's branded in like your website's brand, and so maybe it can be playful, it can have the right tone of voice that's matching your business, and I feel like that's another pretty much no brainer, like why wouldn't you do that? That seems like a nice little thing to have. So I would say that that's probably the baseline for your service worker is cache the kind of assets you want to cache anyway for performance reasons and have a custom offline page, if nothing else, just for branding reasons.

Some people have played around with that concept, the Guardian used to have a crossword puzzle on their offline page, which is really nice, and Trivago, the travel company, their offline page is a game. It's like a maze game you can solve while you're offline. So there are these opportunities I guess for little touches of delight there.

Lea Alcantara: Well, I actually have a question in regards to that user experience, because sometimes offline isn't an extended amount of offline. So for example, your connectivity just sucks temporarily. Is it actually a better user experience to kind of have like, "You're officially offline," here's a note saying that you are versus an experience that almost makes this seem like you're still online because it's already been cached?

Like the way I think about it again is like when I travel, but sometimes I could still see some old Instagram shots or old Facebook posts. So Facebook occasionally has this like little line basically saying you're currently offline. You can still post, but then it will post when the network is back on, but like other apps or other websites don't have that, but then it still seems like it's online. Is there like a pro or con for the user experience over like what is best? Like can a service worker time the message, for example, like if this is more than a minute of offline, then show the offline page because it's assuming that you're going to be offline for a long time versus intermittent connectivity issue?

Jeremy Keith: Yeah, this is tricky one. I don't think there is a right or wrong answer. I think it's another it depends, which is always the answer in web development. Interestingly, the API for the sort of checking, is the user online or offline, that's separate from service workers. This is like — I forget what it's called. Some JavaScript API you can do just from within a web page, so even leaving service workers out of it. If someone is on a page and you have the ability to detect that their internet connection drops, should you do something with that? So do you do show a little message at the top, a little banner at the top saying, "Oh, you're offline right now," and the answer again is, "Well, it depends.

I mean, if I'm in a middle of reading an article then maybe don't show me that because I don't care. I'm still reading the article." But yeah, if I'm filling in my taxes, maybe I do want to know or maybe you use that API to disable certain functionality while you're offline. I think it's kind of the way that Slack works, that if you lose internet connection, you literally can't type into the input until the connectivity comes back. Maybe that's what you need to do.

Or on the other extreme, there is this API called Background Sync, which doesn't have as much wide support as service workers, but it's getting there, where you can pretty much just let the user carry on doing what they're doing even though you're offline and be saving up all the data they're inputting and be putting it to one side, store it locally, and then syncing up with the server later when the internet connection is back. But here's the thing with the Background Sync API, where you can sync up with the server later even if the browser isn't open, which is really powerful, so really, in that kind of situation, you could kind of decide, "I'm not even going to inform the user that they're offline and then in the long run, it doesn't make any difference then. They're going to be able to accomplish what they want to do. They're going to get the data to the server so I'm not going to do it."

But there's all that gray area in between, it's like filling in a form. Does the user want to know that maybe that this might not complete because they're offline versus I'm reading an article where I was like why do I care if I'm offline or online, I'm reading an article. Yeah, this is really interesting because all of these questions aren't really technical. They're all about the user experience, and that's where I think it gets really interesting with things like service workers, Background Sync, all these APIs, it's not really about the technology, it's about the experiences and asking you, "What is the best to be offering here or what's the right thing to do in this situation?" And a lot of times we're trying to second guess what our users want. We're kind of making these decisions on their behalf. It's like, "Okay, I think if the user is offline, then they would appreciate this experience." It's interesting, that's where I feel like there isn't a right or wrong answer to lots of this. It depends on the site. It depends on the user. There are so many variables.

Emily Lewis: I think it even points back to what you were saying earlier that when putting together, building a service worker for your site, the actual build of it, the technical isn't as challenging as deciding what it is you want it to do for you, that you have to take into account whatever your business's goals are, but also that user experience, and that process I think is I find that, as I said, I'm trying to put a service worker on the new site that we're going to launch in a couple of weeks, and that's where I'm spending my time. What should I have it do? Like I understand what it's doing now. I built one, but what should it actually be doing?

Jeremy Keith: Yeah, that's a tougher question to answer, isn't it? Right?

Emily Lewis: [Agrees]

Jeremy Keith: I mean, I think JavaScript for sure, and I would go as far as to say web development in general is effectively an active translation. With CSS you're trying to translate something visual into what the browser understands. With JavaScript, yeah, it's literally a script. Do you know what I'm saying? We write a script for the theater or for a film. It's like this happens, and then this happens.

Emily Lewis: [Agrees]

Jeremy Keith: But with things like ifs and loops and all this kind of stuff, but you have to start with figuring out what it is you're trying to do and then you translate it into whatever language is appropriate; CSS, JavaScript or whatever.

Emily Lewis: [Agrees]

Jeremy Keith: But yeah, you can't jump over that first step, which is what am I trying to accomplish here? I wrote this script in English, what would it say?

Emily Lewis: I think that also leads nicely to part of the book that you talked about towards the end, but kind of what I perceived as like best practices. Obviously, everyone is going to have their own rules that they want to build into a service worker, but that there are some things that we can do as responsible developers, for example, like cleaning up caches and stuff.

Timestamp: 00:40:09

Jeremy Keith: [Agrees]

Emily Lewis: Can you describe that a little bit more?

Jeremy Keith: Yeah, so service worker is kind of a bit of an umbrella technology where you get access to these other APIs. The Fetch API I mentioned, that's where you make request out to network and the Cache API, so the Cache API is what browsers have had under their hood for years. This is how they do the HTTP cache, but we've never had access to it as developers.

Emily Lewis: [Agrees]

Jeremy Keith: And now we do through service worker, which is great, and so with the Cache API, you could get to do what browsers have been doing for ages, which is, "I'm going to create a cache, I'm going to put files in it, take files out of it, look files up, great." But as well as getting that control and power, you also now get the responsibility of taking care of that cache, cleaning things up, so the cache doesn't get out of hand. So there's kind of, yeah, I guess kind of cleanliness and hygiene involved here.

Emily Lewis: [Agrees]

Jeremy Keith: It's like, well, you don't want the cache to just fill up and fill up and fill up over time, so the next time someone hits your website, maybe you'd take a little time out to clean up those caches, which all happens in the background. It's not going to slow down the user experience or anything, but it is up to you. Again, you don't get anything for free here. You've got to think about what's the right thing to do and then go through it, and yeah, cache hygiene, I guess, is one of those things. It's like, are you cleaning up your old caches? Are you making sure there aren't too many files in there? Yeah, just being a responsible citizen.

Emily Lewis: You described sort of that minimum experience with service workers and even mentioned the Guardian as an example of something that they had done, just an offline page, but have you seen a real-life example where a service worker kind of creates a really robust offline experience for someone?

Jeremy Keith: Oh, I mean, so you can take it to the extreme. So yeah, the lower end is you just have a custom offline page. So if someone visits your site and they tried to access something and they end up with the custom offline page instead, that's the baseline one. Now, there are all sorts of places you can go from there. So let's say we're talking about a content site, assuming it's something people are browsing around and reading articles, let's say. Well, as they're browsing around, you've already fetched that article from the network. You could decide with your service worker, "Okay, I'm serving this up to the user, but you know what, I'm also going to keep a copy of this and put it in this cache over here called pages or articles," whatever you want to call it, and so every time someone is reading an article or visiting a page, a copy is getting squirrelled away into the cache.

Now, let's say they go offline and then they tried to hit an article they've already read it, you could say, "I'll just serve it straight from the cache. I've tried to the network to get the freshest version, but the network is unavailable. All right, I'll try it from the cache, and if there's a copy there, show that. Don't even tell them you're offline. Just show them the cached version."

But then you could have a third step, which is, “Okay, you tried the network, you tried the cache, and neither of those worked. Last ditch attempt here, we’re just going to show the custom offline page.” So you do that, so you tried the network, that didn’t work. You tried the cache, that didn’t work. You show your custom offline page. But even here on the custom offline page, you could give it some oomph. You could say, “Okay, now sorry you’re offline. Sorry about that.” But you could then at that point show, “Hey, listen, these are these pages that you already visited, and I’ve got copies of them, so if you want to read any of these, even though you’re offline, go for it.”

So that’s a pattern I’ve used on my own website, on my own blog. So if you lose your internet connection while you’re surfing around adactio.com, as long as you’re hitting pages you already visited, you won’t even know that you’re offline. Until when you tried to visit a page you haven’t yet visited that you’ll get the custom offline page, and then on that custom offline page, it says, “Here are the pages you’ve already visited. Knock yourself out.”

And there’s a few other people have that pattern on their website. Sara Soueidan is doing them on her blog and Ethan Marcotte as well, and that’s pretty good. Again, this kind of goes back to we’re trying to guess what the user wants here so I’m thinking this would be a good experience, but I’m only showing them stuff they’ve already read. You can take it up to the next level, which is that while they’re online and browsing around your site, you could have an interface element, which is, “Save this for offline.” So now instead of you trying to guess what the user wants to save offline, the user actively tells you, “I want to save this for offline.” It’s in the same way you do with InstaPaper or Pocket or any of these kind of things, except it’s like a personal one just for your site, and so then when they try to visit your site, it’s offline, you’d give them the custom offline page, but instead of saying, “These are the articles you’ve visited earlier,” you can say, “These are the articles you explicitly said you wanted saved for offline.”

So it becomes much more functional then I think. It's much more like the user can plan, "Oh, I've got a plane ride ahead of me. I'm going to go to this website, save some articles for offline. I'm going to read them on the plane." It's much more I guess in the user's hands there.

So at each level, you're kind of taking it up and you're adding a bit more complexity probably, but also really pushing the offlineness of the experience, and in the final level you could go to is, is it possible to just never touch the network at all where you can literally be offline first? It's where your default response isn't to go, "Try the network and then try the cache," but to go, "Try the cache first for everything, and only go to the network as a last resort."

Now, that's not going to work for up-to-date stock market information or sports results, stuff that needs to be fresh, but for some kind of content, more evergreen content, that actually might be the best approach to take, and what you can also do is you could pre-cache stuff, so you could say, "Let's say it's fairly small website. Let's say it's just kind of a brochure website, and so you've got less than ten pages. The user hits that first page. The service worker gets installed. You could have the instruction to the service worker and say, "Grab these other nine pages and put them in this cache." And from then on, no matter what they request, it's going to come from the cache. It's going to be super speedy and so this isn't just about the offline experience, but this is also where I've got a perfectly fine internet connection, but it's still going to be faster than anything I could have experienced going to the network because it's coming straight out of the cache.

So there are all these different possibilities and all these levels you can choose to do, going from custom offline page all the way up to offline first.

Lea Alcantara: So I'm curious because with all these different choices and you mentioned that we're kind of guessing what the user needs. Now, again, I'm probably thinking about this in an online first kind of mindset, but with Google Analytics, for example, you need to be online so they can track where you're clicking and you can track the efficacy of certain pages. How can you track an offline

experience and see which experience is better whether you should give them just an offline page with certain lengths or a fuller experience?

Jeremy Keith: Well, I'm not sure if this is what something like Google Analytics would help answer that question. I think this is more of a testing with real users.

Lea Alcantara: User testing.

Emily Lewis: [Agrees]

Lea Alcantara: Yeah, okay.

Jeremy Keith: Yeah, I think that's much more user testing. It is possible to get analytics working. Even offline, there are ways of pinging out to Google Analytics or you can imagine using something like Background Sync to queue up those pings and then hit the Google server when it does. .

Emily Lewis: [Agrees]

Lea Alcantara: Okay.

Jeremy Keith: To be honest, analytics is not my forte and I even find them slightly distasteful slightly.

Emily Lewis: [Laughs]

Lea Alcantara: It's fair. [Laughs]

Jeremy Keith: Creepy shall we say.

Lea Alcantara: [Laughs]

Jeremy Keith: But in terms of user testing or usability testing, I should say, and figuring out by watching users use something is gold dust. I mean, that's really how you get to the bottom with some of these questions, I think.

Emily Lewis: That kind of leads into the question I was going to ask, which is as a developer, this sounds ground, and Lea, I know that you do a ton of caching for our clients on the CMS's end, so we are already on board with encouraging our clients to have speedier sites in general, but how do we translate this as an option for a client? Is it something that we need to "sell" or educate them on or is this something that we, as developers, should be building as part of our sites in the first place, and it's not even something that the client needs to know about?

Jeremy Keith: That is such a good question. I think it was a matter of priorities here, and it sounds like you've got your priorities straight already, which is that performance is something you do by default, and that's something to bear in mind with service worker. You might think, "Oh, I want to try out this cool service worker technology." It's like get your performance house in order first, right?

Emily Lewis: [Agrees]

Jeremy Keith: Because the service worker will not help with that first visit to the website. You know, when someone visits your website for the first time, it does not matter whether their browser supports service workers or not. There can't be a service worker installed until they've already visited, so you have to make sure that first time experience is as fast as it can be. So if I was going to prioritize improving the performance of my website or adding a service worker to the website, I would start with improving the performance, and then move on to, "Okay, next level, let's add the service worker."

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: So it is definitely get on top of what you would already do. Now as to whether you treat it as business as usual, I mean, if you swallow the cost of that, that's a really interesting one. I would like to think that over time it would become that. Do you remember when responsive web design first came around and we had these discussions as well?

Emily Lewis: [Agrees]

Jeremy Keith: It's like do we charge more for making it responsive, but then at some point, it just became the default.

Emily Lewis: [Agrees]

Jeremy Keith: It just became "That's how we do things. Well, how could we not make it responsive?" And I think the more you use service worker, the more of a feeling you get for it, the easier it is to just treat it as business as usual. It's like, "Oh, yeah, we'll add a service worker, no problem."

Emily Lewis: [Agrees]

Timestamp: 00:49:54

Jeremy Keith: Now, the first time you're adding service worker to the site, it's going to take you a long time presumably because you're learning, and that's no different to the first time you tried to make a site responsive, it took way longer than it does today because we just got used to it when it became normal, became business as usual. So I think over time, it probably would become just

something you do without even mentioning it to the client potentially. It would just be like something you provide by default in the same way you don't say that making a favicon is extra.

Emily Lewis: Right.

Lea Alcantara: [Agrees]

Jeremy Keith: It's just that you do it. It's just on that checklist of things you do, and maybe a service worker is there, at least as a minimum kind of service worker.

Emily Lewis: [Agrees]

Jeremy Keith: But yeah, it's always going to be tricky at first, and as to what you are trying to sell this in, that's a really good question. I think it depends on the site again and it depends on the client. For some things, you could say like, "Well, we'll definitely add in the service worker that caches CSS and JavaScript and icons. We're not going to ask for permission for that." But if we have more time and more budget, we could really craft an interesting offline experience around that idea of letting the user explicitly cache articles, stuff like that.

Lea Alcantara: [Agrees]

Jeremy Keith: So that would take more time probably so it probably takes more budget and then, yeah, you probably want to have that discussion with the client, explain the pros and cons, describe the user experience benefits and see whether they want to go for it.

Emily Lewis: Interesting. So the last part of your book, you get into progressive web apps, and I have to admit that they've been around for a while and people have talked about them, but it wasn't until I've read your book that it became crystal clear for me what they were, and they're not, in my

mind at least, very complicated, and so can you describe what a progressive web app is and a service worker fits into that?

Jeremy Keith: Sure. I'm going to give you what I think is the cold heart facts of what a progressive web app is, which is this, progressive web app is a website that is running on HTTPS, has a service worker that serves at least some kind of offline page, and has a manifest file, and that's a JSON file with metadata in it. It's like, "Here are some icons, here's the name of the website." That's it.

Emily Lewis: [Agrees]

Jeremy Keith: From a cold heart facts point of view, that's what a progressive web app is, and the reason why you're quite really confused or like trying to figure out what this is, because if you try to actually look up what a progressive web app is, no one is going to say that. No one is going to say, "Oh, it's these three things."

Lea Alcantara: [Laughs]

Jeremy Keith: They're going to describe it in almost metaphysical terms.

Lea Alcantara: Yes. [Laughs]

Emily Lewis: [Laughs]

Jeremy Keith: It's like, "Wow, it's an experience so rich where you provide the user with the seamless…" And there are all these adjectives around rich and performance, blah, blah, blah, and it's like you sound like you're just describing a good website.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Like what are the bits that make it specifically a progressive web app? And I've boiled it down to those three; HTTPS, service worker, and manifest file, and also interesting, those are three testable things, you can test for the existence of those things, and then give it a check mark and say, "Yes, this is a progressive web app.."

Emily Lewis: [Agrees]

Jeremy Keith: Now, it's true enough that you could have all three of those things; HTTPS, service worker, and manifest file, and still build a terrible website. That's like inaccessible, isn't performant, it is isn't responsive, and it's probably right to say that then it's not really progressive web app, but I would say, "You know what, when we're talking to developers at least, let's use the language of development and say, 'These are the technical criteria of what a progressive web app is.'"

Now, if you're talking to a client, if you're talking to a marketing department, maybe yeah, then you want to use the adjectives and talk about experience and richness and performance and all this kind of stuff because they don't care about the technical details of this stuff.

So I think you need to adjust the language depending on who you're talking to.

Lea Alcantara: [Agrees]

Jeremy Keith: But what that means is, yeah, just trying to get a straight answer to the question, what is a progressive web app, is way harder that it really should be, you know?

Emily Lewis: [Agrees]

Jeremy Keith: I mean, I always like that with responsive web design, Ethan provided like just this, he set the boundaries, "These three criteria are what make it responsive."

Emily Lewis: [Agrees]

Jeremy Keith: And even at the start, people are like, "Well, yes, but I mean, there's a bigger sense of responsive. Is it really truly responsive?" And he goes like, "No, these three things, that's what defines it."

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: And I always appreciated that; that he kept the boundaries nice and clear, and I'm kind of trying to do the same thing for progressive web apps. I'm trying to encourage people to think that way. It's like there are three testable technical criteria for something being a progressive web app.

Emily Lewis: And someone choosing to have those three criteria in place in an intentional matter, how do I phrase this? For example, we have HTTPS on our current site and I'm going to try service worker when we launch with the new design, and then maybe I'll put a manifest together, but that's just putting the pieces together, what's the bigger picture of having a progressive web app?

Jeremy Keith: Right. So exactly, under those pieces, you can do step by step. You don't have to do them all at once, like what you're talking about there, you're going, "At some point, you do this and then you'll add on the manifest." Like I see you have a question, "But why? Why would I do that?" So there are security benefits for HTTPS. There are performance benefits and user experience benefits for service worker, but what do you get by putting all of this stuff together, having the tick

mark for all three of those things? Well, for one thing, on Android, Samsung Internet, and Opera will try and alert the user that this site is they think a quality website because it fulfills those criteria being a progressive web app, and encourage the user to affectively install that like as though it were a native app.

Emily Lewis: [Agrees]

Jeremy Keith: So the browsers do it differently. Some have a little popup. Some have something in the browser. Chrome shows a new icon, but effectively you're bookmarking a website. There's nothing new about that. You're bookmarking a website, but on mobile, it's phrased as "Add to Home Screen."

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: And then in that web app manifest, you can specify how this website should open when it's open from the home screen, and that means you can specify, "Well, don't show any browser crumb, but open it full screen. Open it with a minimum amount of browser crumb." And so you really are getting to an experience for the end user that would feel very, very similar to native, and this goes back to again altering the expectations of what users have come to expect of the web, and so now somebody is looking at their home screen and there's a bunch of icons. Some of those icons will launch a native app, some of those icons will launch a website, and the experience shouldn't feel any different if you've built it the right way.

Emily Lewis: [Agrees]

Jeremy Keith: Really, a great email recently. I run this website about Irish music and it is a progressive web app, and I got this email from this person who had recently switched tablets, and so he got a new tablet, and on his old phone or tablet, as he had described it to me, he had this website or he had this thesession.org thing that he had it installed and he was wondering how he got it installed on his new tablet, and I was like, "Oh, this is great. He doesn't even realize that it was a website. He was thinking it's installed like an app."

Emily Lewis: [Agrees]

Lea Alcantara: [Agrees]

Jeremy Keith: So I wrote and told him because he was looking in the app store for it, and I was like, "Well, actually, just go to the website on your browser and do the "Add to Home Screen" thing and now it's effectively installed." I mean, it's been installed the whole time just by visiting the website.

Emily Lewis: [Agrees]

Jeremy Keith: And that just give us a lovely feeling to know like, "Oh, from the user's perspective there, they didn't even remember whether they got it in the app store or whether they got it by "Adding to Home Screen" from a website. The experiencing of using the thing was indistinguishable.

Emily Lewis: [Agrees]

Jeremy Keith: And it will be great if we could get to that where the user just doesn't care whether they got it from an app store, whether they got it from the web, it feels the same, but because it's the web, we get to keep all these great superpowers on the web so when you need to roll out an update, you don't have to submit something to an app store, right?

Emily Lewis: [Agrees]

Jeremy Keith: You just update it. When you want to publish something, you just publish something. You don't have to ask anyone's permission, and you've got URLs. The installation is literally visiting the URL.

Emily Lewis: Yeah, it's very, very cool. Oh, we could definitely talk about this a lot longer. [Laughs]

Lea Alcantara: [Laughs]

Emily Lewis: But we do need to wrap up. Other than your book, which I really do recommend for our listeners, as I said earlier, it was a pretty quick read and it was very – I hate this – it's so "readable." It sounds like people who say wine is "drinkable."

Lea Alcantara: It was funny.

Jeremy Keith: [Laughs]

Lea Alcantara: Yeah, funny, too. [Laughs]

Emily Lewis: I think that's the other thing that I like about your writing, Jeremy, there's this level of humor and also a little of self-deprecating humor, like everyone is probably like, "Oh, Jeremy is so great in all this," and you'll admit that you have your own weaknesses and it sort of makes you feel like, "Okay, I can do this, too, kind of approach to building something." [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Well, definitely, I didn't want to assume any prior JavaScript knowledge with this book. There are plenty of resources out there for you learning service workers, but a lot of them assume that you already a lead JavaScript ninja.

Emily Lewis: [Agrees]

Jeremy Keith: And I didn't want to assume any JavaScript knowledge because I think there is a big audience of people who they know their HTML, they know their CSS, they know their content, they know design, but JavaScript is just intimidating because it's a programming language, but if you want to use a service worker, there's no getting around past the JavaScript, but that doesn't mean I have to assume everyone already knows JavaScript, and it was a real challenge to do that, but I really wanted to make sure that it was accessible to that audience of people, that they didn't feel left out by that.

Emily Lewis: [Agrees]

Jeremy Keith: And actually, one of the ways I kind of kept a check on myself was I had two technical editors, and one was Jake Archibald who literally wrote the service worker's spec, like he knows everything there is to know about service workers. So he was there to catch any technical things I said that were complete nonsense and say, "That's false, fix that." But my other technical editor was Amber Wilson and she's new to web development.

Emily Lewis: [Agrees]

Jeremy Keith: She's a junior developer. She knows her HTML. She knows her CSS. I met her at Codebar, which is a great place for people learning front-end development, but she's just starting out with the JavaScript, at least she was when I was writing the book. She's quite a ways along the road

now, but she was there to catch anytime I was assuming too much, anytime I went too fast or anytime I skipped over something.

Timestamp: 01:00:01

Lea Alcantara: [Agrees]

Jeremy Keith: So I had this balance with Jake and Amber of trying to satisfy the technical criteria, this is technically correct, but also trying to satisfy the criteria for "Is it understandable? Is it readable?"

Lea Alcantara: [Agrees]

Emily Lewis: I think you struck the balance well. I was really not surprised because I've been reading you for a long time, but it was really pleasant that the minute I finished it, I was like, "Oh, I'm just going to go build this." Like I just knew I would be able to do it. [Laughs]

Jeremy Keith: So this is the best feedback I get about the book. It's nice when people say, "Oh, I read your book and I liked your book." That's nice. But when people say, "I read your book and now my website has a service worker," that's the best, right?

Emily Lewis: [Agrees]

Jeremy Keith: That's the best feedback ever.

Emily Lewis: So listeners, definitely check out *Going Offline*, but Jeremy, do you have any other resources you could recommend for our listeners who want to learn more about service workers and offline experiences?

Jeremy Keith: Yeah. I mean, I get a lot of benefit from people explaining things in their own words, and so there are a lot of great blog posts out there by people who have added service workers to their own websites, and then described that process. I list some of them in the book and I made a blog post with those resources. I'll take out that blog post, but I know of people like Ire Aderinokun and Sara Soueidan and Una Kravets, Ethan Marcotte, Jason Grigsby. These people have sort of gone through the process of "Well, here's I added a service worker to my website," and I just find those the best when people are just sharing what they learn. So yeah, those kinds of resources I find great.

On the subject of progressive web apps, I know I'm not supposed to say this, but there will be A Book Apart book specifically about progressive web apps, too.

Emily Lewis: Oh!

Lea Alcantara: Oh!

Jeremy Keith: So it's going to be like a one-two punch with the *Going Offline* book. If *Going Offline* is the technical side of "Here's how to make a service worker, this book will be about kind of business benefits and how to sell it in the user experience questions around progressive web apps.

Emily Lewis: Oh, awesome.

Lea Alcantara: [Agrees]

Jeremy Keith: So keep an eye out for that one.

Emily Lewis: Excellent.

Lea Alcantara: So Jeremy, final advice for our listeners diving into service workers for the first time.

Jeremy Keith: Oh, well, have fun with it, and also think about what you want to accomplish first before you start writing any code, and do treat it as an enhancement. It's not a silver bullet. It's not going to make your website magically amazing if it wasn't a good website to begin with. Think of it as a way of taking an already good website, you've already got an accessible, performant website, and now you're just adding that icing on the cake with service worker. I thin, that's the way to approach it.

Lea Alcantara: Sounds good. Well, that's all the time we have for today, but before we finish up, we've got our rapid fire ten questions so our listeners can get to know you a bit better. Are you ready?

Jeremy Keith: Ready as I'll ever be.

Emily Lewis: [Laughs]

Lea Alcantara: All right, first question, what's your go-to karaoke song?

Jeremy Keith: *Wichita Lineman* by Glen Campbell.

Emily Lewis: What advice would you give your younger self?

Jeremy Keith: I would inform my younger self that we've somehow invented time travel and that they shouldn't mess with the time line because it will be disastrous.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Destroy the machine as soon as it's invented.

Lea Alcantara: [Laughs]

Emily Lewis: [Laughs]

Lea Alcantara: Amazing. What's your favorite PG-rated cuss word?

Jeremy Keith: Oh, I'm not sure if it is PG.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Emily Lewis: You can say it.

Jeremy Keith: Cockwomble. Is that PG?

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Emily Lewis: I don't even know what it means, but I'm going to start using it. [Laughs] Who is your favorite superhero?

Jeremy Keith: Wonder Woman.

Lea Alcantara: What is your favorite time of the year?

Jeremy Keith: Summer.

Emily Lewis: If you could change one thing about the web, what would it be?

Jeremy Keith: Oh, gosh, permanence.

Lea Alcantara: [Agrees]

Jeremy Keith: The fact that we have so much link rot and things disappear over time.

Emily Lewis: [Agrees]

Jeremy Keith: I would change that.

Lea Alcantara: What are three words that describe you?

Jeremy Keith: Oh, this is hard, indecisive.

Emily Lewis: [Laughs]

Lea Alcantara: [Laughs]

Jeremy Keith: Procrastinating. Incompletist.

Emily Lewis: [Laughs] How about three words that describe your work?

Jeremy Keith: Mediocre, long lasting, and fulfilling.

Lea Alcantara: What's your favorite meal of the day?

Jeremy Keith: Dinnertime.

Emily Lewis: And last question, coffee or tea?

Jeremy Keith: Coffee in the morning, tea throughout the day.

Lea Alcantara: Very cool. So that's all the time we have. Thanks for joining the show.

Jeremy Keith: Thank you. It is a pleasure.

Emily Lewis: In case our listeners wanted to follow up with you online, where could they find you?

Jeremy Keith: My website is very much the place to go, adactio.com. I'm also on all the social networks and stuff, but those are all just copies of stuff I send through my website. So if you want to go to the source, it's my website.

[Music starts]

Emily Lewis: Thanks again, Jeremy. It was so good to talk with you.

Jeremy Keith: It was my pleasure. It was great.

Lea Alcantara: CTRL+CLICK is produced by Bright Umbrella, a web services agency invested in education and social good. Today's podcast would not be possible without the support of this episode's sponsor! Many thanks to Luke Stevens' toku!

Emily Lewis: We'd also like to thank our hosting partner: Arcustech.

Lea Alcantara: And thanks to our listeners for tuning in! If you want to know more about CTRL+CLICK, make sure you follow us on Twitter @ctrlclickcast or visit our website ctrlclickcast.com. And if you liked this episode, consider donating to the show — then give us a review on Stitcher or Apple Podcasts or both! Links are in our show notes and on our site!

Emily Lewis: Don't forget to tune in to our next episode when Travis Gertz returns to the show to talk about editorial designs, specifically applying editorial design with Craft CMS. Be sure to check out ctrlclickcast.com/schedule for more upcoming topics.

Lea Alcantara: This is Lea Alcantara …

Emily Lewis: And Emily Lewis …

Lea Alcantara: Signing off for CTRL+CLICK CAST. See you next time!

Emily Lewis: Cheers!

[Music stops]

Timestamp: 01:05:31