EPISODE 1368

[INTRODUCTION]

[00:00:00] KP: According to builtwith.com, more than 10 million websites are powered by React framework. Of the top 10,000 sites by traffic, 44.7 of those are built with React. This JavaScript framework is capable of powering a wide array of modern applications, and remains fairly beloved by developers that use it.

In this episode, I interview Kent C. Dodds, software engineer educator. We discussed Kent's journey learning React and keeping up with the changes that's taken on as the framework evolves, and how those lessons culminated into his epic React course.

[INTERVIEW CONTINUE]

[00:00:39] KP: Kent, welcome to Software Engineering Daily.

[00:00:42] KD: Thank you so much. It's a pleasure to be here.

[00:00:45] KP: To kick things off, could you tell me a little bit about your story becoming a developer?

[00:00:50] KD: Yeah, sure. So when I was like a preteen, my best friend was really into software programming. And he like wrote a server in C, and I had no idea what that meant. But it sounded interesting. We were going to build a game together and stuff and make millions of dollars. And so he tried to teach me some software. And he taught me what Boolean was. That made sense. He taught me what a number data type was, and that made total sense. And then when he started trying to teach me about what a string was, I just couldn't figure it out. Like I didn't get past that.

So I did do a little bit of HTML and CSS in those years. But it wasn't until I got into college and I actually took a programming class that I realized how much I didn't like developing software. My first two classes, I did okay in them, but I had like a Java class. And yeah, just did not enjoy that a ton, because it was just too theoretical. Like, let's talk about linked lists and stuff. And I was still trying to figure out who cares. And then I had a computer systems class. And that also was really low-level. Like we wrote a program in zeros and ones. Like that was not fun to me. It was interesting, but the thought of sitting in front of a computer all day was like, “Yeah, doing that was not fun.”

So it wasn't until I went on a mission for my church and then came back and took another path. I was going into information systems. And I took another programming class, and I had a job where I was supposed to manually upload a bunch of videos to YouTube. So I had to download videos from one server, upload them to YouTube. And because I was taking a programming class, I could automate some of the process. And I did, and it worked. And it just, like, blew me away how software could be practical.

And so all of a sudden I went from, like, programming is not for me at all. To programming is super fun and interesting. And then eventually, when I discovered JavaScript, and found how fun that ecosystem is, everything kind of fell over from there. And I've just loved it ever since. Maybe that was about a decade ago. No, no. Yeah, actually, maybe eight years ago. I was still in school at the time. And yeah, it's just been super fun ever since.

[00:03:00] KP: At what point did you discover React?

[00:03:02] KD: That was the beginning of 2015. I think it had been out for about a year by then. And a bunch of people that I respected and looked up to, and my friends seem to think it was really awesome. I was way into AngularJS at the time. And like I had a podcast, and I was a contributor to AngularJS core. And so I was really into it. But I took a look at React and I was very fascinated by some of the ideas there.

It wasn't until the end of 2015 that I actually switched jobs and got a React job at PayPal that I really jumped into React. And so, yeah, it's been about six years with React now. It's been awesome. Really, really happy with React.

[00:03:44] KP: React has evolved quite a bit. When I started, there were no Hooks. And I think even the way I manage routes has changed three or four times. What's that experience been like keeping up with this evolving framework?

[00:03:55] KD: Yeah, that's actually one of the appealing things about React to me is that it can evolve like that. With Angular JS, it was – Well, I mean, backbone never really did evolve. That's where I kind of started with my frontend frameworks. And so it's pretty much what it's always been. And so almost everybody's moved off of that. And AngularJS, they didn't really evolve. They created a new framework and kept the name.

Ember has done a pretty good job of evolving and changing as requirements have changed. But they've really struggled to get mindshare. React has done a phenomenal job focusing on like their core competency of the Vue, and state management for the UI, and then letting the community kind of collaborate together and innovate. And I think that's one of the reasons why React is so popular is because the community can innovate so well around different ideas. We don't have to wait for the framework team to kind of figure different things out. We can kind of figure things out on our own. And now you've got a team of hundreds of thousands of people working on solving these problems.

And while that does lead to a little bit of churn, it allows React to stay focused on what they do and what they do really well, and enables them to evolve. Because yeah, I also started with react.createclass, and you had mixins for code sharing. And then went through the class syntax. And each one of these had their own set of tradeoffs and problems and just kind of decisions that they made. And then Hooks came around and that has just been a game changer. So much, in fact, that Vue has adopted the same pattern as well. And other frameworks are trying to adopt the same concept of Hooks, which is just a really great model for code reuse. And so I've been really happy with Hooks for the last – Oh gosh, I think it was three years ago that they were announced, maybe four. And I haven't written a class component in years now. So it's pretty much stuck on React.

And yeah, the router has evolved as well. React router is kind of the de facto standard. And that, a lot of people give Ryan and Michael Flack for their evolving router. But it actually hasn't changed a whole lot. I mean, it has changed a fair amount, but not super rapidly. Like the last change was several years ago. They're coming out with a version six that's just going to knock everybody's socks off. And I'm really looking forward to that. It's kind of based on the work that they've been doing with Remix, which I'm also super, super excited about.

So yeah, the React ecosystem itself has evolved. But I think that's one of the things I like about it is its ability to evolve as new ideas come up. And we can keep most of the things that we understand and most of the things that we've learned. That was one of the things that bothered me when I left Angular. The Angular community was so much of the knowledge that I had accumulated was useless to the rest of the skills that I needed for the rest of my development, whereas with React, I spend most of my time learning JavaScript or TypeScript. And those skills only helped me in other areas as I'm working with React.

[00:06:58] KP: At what point did you decide you want to not just write code but teach other people how to write code?

[00:07:04] KD: Oh, teaching for me is always inseparably connected to learning. Even from like at a very young age, when I was in church, and everything I've always been encouraged to teach. Like in church, we speak in front of our peers and everything. So it's been very natural to me to teach. The first coding thing that I ever taught was actually the first meet-up I went to. I looked on the meet-up page and saw that they were looking for a speaker. I'd never been to a meet-up before. So I didn't realize that meet-ups are literally always looking for speakers. So I thought, “Oh, I guess I'll help out.” And I spoke about a library that I had written. And then I also put together an Angular a workshop for my classmates when I was in school. So yeah, this whole time that I've been – And actually in that first class, that first programming class, my first year in college, I volunteered to be a tutor as well.

And so for me, teaching is the way that you solidify the things that you've learned. And so you don't really – Like you have to be able to articulate what it is that you've learned before you can really say that you've actually learned it. And so as far as like when was it that I decided I wanted to teach for money, that was when I realized that I could, I guess. When I was actually still in school, and I'd given a meet-up talk and that was recorded and put on YouTube. And John Lindquist, the creator of egghead.io, saw that and said, “Hey, Kent, we want you to turn that into a course for us.” I think I was instructor number like 12 or something on Egghead. So it was pretty early days back in, I think, 2014 probably. And yeah, and I made a course and started like paying my mortgage with this side hustle. And so I realized, “Oh, wow! Like not only do I get to teach to solidify my understanding of what I know, or what I think I know, but I also get to make a pretty good like side income while I do it.” So I've been teaching for basically ever.

[00:08:59] KP: Very cool. What sort of content will listeners find it epicreact.dev?

[00:09:06] KD: So epicreact.dev is clearly a React platform or course, but it's certainly different from what you're thinking when you hear me say the word course. You're probably trying to compare it to something like Udemy, or an Egghead course or something like that. It's nothing like that. Basically the best way to think about it is like it's a semester class, like a three credit class at a university, where like the expected completion time is around 14 weeks because there's just so much content there. There are around 350 videos. And as much time you spend watching the videos, you actually spend two to four times more time on your keyboard working through exercises. So it's very exercise-focused.

And to give a little bit of backstory on why it exists, I've been teaching React to pretty much since I started using it. And as I've been teaching workshops, slowly I started with one workshop and then realized I needed to split it into two and then more as I developed more content and wanted to focus each workshop on different aspects. And so I have a fundamentals workshop, and then a performance workshop. And we've got advanced patterns, and advanced Hooks, and basic Hooks and actual, like, let's now apply these things to something practical and build a real app. And that's like four workshops. So yeah, there's just a lot of content there. And also testing.

And so I was delivering these workshops. But because I don't scale very well and my time is valuable, it was really inaccessible to people because it was so expensive. And then there's also – Like I can't do things at every learners pace. I just have the pace and hopefully it matches closely to your pace. And so for those reasons, I realized that I wanted to scale this beyond myself so I could reach more people, help more people to learn this so they can make the world better. And so once I solidified the material by delivering these workshops dozens of times, I decided to record it as if I were giving a workshop to just a single individual. So I'm talking to the camera and talking to you and saying, “Hey, here's what we're going to be doing for this exercise," and go and do it. And then I work through the solution with you. And yeah, that's what Epic React is. We also have, I think, it's 10 or 12 episode podcast with React experts. And yeah, a bunch of other cool and interesting things both free and paid on epicreact.dev.

[00:11:32] KP: Well, what are some of the prerequisites? How much should a beginner know before they take on that course?

[00:11:39] KD: Yeah, I have a blog post called the JavaScript you need to know for React. So I make an assumption that you are experienced with JavaScript, and maybe a little bit of HTML and CSS, but mostly JavaScript. And then if you have a good understanding of the JavaScript features that I list in that blog post, then you should be pretty solid. And it's nothing super crazy. Like we don't do generators or anything like that. But we're using default parameters. And we're using destructuring and default values there, and promises, and object spread and stuff like that.

And so yeah, just some fundamental JavaScript features that most developers should know and be comfortable with. And then, yeah, having some experience with React is useful, but not necessary. I do have a beginner's guide to React course that is free on Egghead. And so often people will take that first to kind of get a feel for what React is all about. And that is not exercise-based at all. That's like sit back, take notes sort of experience there. But if you go through that and you're like, "Okay, yeah, this sounds interesting. I want to really learn React. Then Epic React is the next step.

And yeah, you don't have to have any experience developing for the web either. But of course, having some experience, and mostly exposure to the problems that we're trying to solve is really helpful for being able to understand the problems that we're solving throughout the course. And so the more experience you have, the more you can fall back on when you're thinking about what you're learning. But I wouldn't say that precludes you from learning from the experience. And part of the learning process also is actually applying the things that you're learning so you have a good understanding of how things work. And this is one of the reasons why it takes so long to go through the material, because you, you technically could watch all the videos and get through the exercises if you just did it every single day for two weeks. But I certainly wouldn't recommend that because your brain needs breaks. And you need to apply what you've learned in an app that you're working on. Or you just kind of interleave some of the final workshop where we're building an app together. Interleave some of that with what you're learning. And that's actually in the recommended schedule that I have for learning Clubs for people who want to go through this content together. I try to not assume much more than just some foundational JavaScript knowledge.

[00:13:58] KP: Let's now switch gears to a different persona. Imagine a seasoned software engineer. They know JavaScript. They know HTML and CSS, but they spent most of their time not developing web apps. So good coder, they're going to be new to some of the concepts of React. Are there common points of frustration you've observed in that path? Things that are counterintuitive at first, or people struggled to learn initially.

[00:14:21] KD: I find that people who have experienced with React, with like old version of React, where you're using classes and stuff, they sometimes struggle with Hooks. Beginners will sometimes struggle with Hooks as well just because it seems kind of magical. I think experienced software engineers should be able to pick up on Hooks. Maybe if you don't have any experience with JavaScript, then some of the way that Hooks works might seem kind of magical until you get a good understanding of how closures work. But I think most seasoned engineers shouldn't run into too many problems, especially if you don't have experience with class components, then you have less to unlearn, which is good. People who have experienced with class components have to change the way they think about components.

I guess the other thing is experienced engineers who don't have experience with components and maybe are more familiar and used to object-oriented programming, there's some adjustment there for thinking in more like functions as a first class citizen. And a lot of that comes down to JavaScript, the language, than it does React. Like probably the biggest surprising thing for people with React is Hooks. And then of course, there's JSX. But JSX is the sort of thing you can learn in like a day, and then it's really not a problem. So it's mostly the Hooks can be a little bit of a challenge for seasoned engineers. And the way that I teach it, and Epic React kind of eases you into that. So hopefully it ends up working on. And so far with the feedback that I've gotten over the last year since I released it has been really positive. And nothing in particular stands out beyond that.

[00:16:03] KP: Well, Hooks are powerful. But yeah, interesting and kind of new abstraction. I can see why you need a course to teach them. I don't know that it's fair to ask you to sum them up in a sentence. But for listeners who might not be familiar with React Hooks or the idea of a hooked, what is it?

[00:16:18] KD: Yeah. So Hook is basically a special function from React that allows you to hook into either state, or side effects, or some combination of the two, for managing state and side effects. That's basically it. So you have a useState hook. And by convention, they're all prefixed with use so that we can have some static code analysis tools that can give us warnings when we're doing things incorrectly. And so we have a use state that allows you to manage some state inside of the function body of your component. And so your React component is a function. It accepts some arguments. And we call those arguments props. And then it returns React elements, which is the JSX. And so you'll return your UI.

And so within the body of that function, you can call these Hooks to manage state and side effects. So if we were building a very simple counter, then you could say react.usestate. You pass your initial value as an argument. So we could say zero. And then what it's going to give back to you is a tuple, an array of the state and a mechanism for updating that state. And you can call it whatever you want. We use destructuring to take the first and second items out of that array, or you could call that count and set count. And then you can render a button with an on-click handler that calls the set count function to increment that count. And so that's managing state. And then you also have a react.use effect, which you pass a callback to. And so every time there's a re-render, so every time your component updates some state, then this callback will get called. And you can synchronize the state of the world with the state of your app. So if we're updating the document title to have, whatever, the value of our count is, then we could say document.title equals whatever that count value is.

And so every time that count value is updated, whenever there's a state change, we call a set count that triggers a state change, and that triggers a re-render, then our use effects callback will get called and we'll be able to update the document title. So use effect for managing any side effects based on any state change. So it's synchronizing the state of the world with the state of our application. It can also be used for requesting data to go and say, “Oh, well, the user wants to now get the chats from this chatroom.” They've changed the chat room that they're in.

Okay, so we'll go and get those chats and then update our own state based on what those chats are. So it's kind of synchronizing the state of the world now with the state of our app in a way. And so that's use effect.

And then we've got a number of other hooks to facilitate those two core competency. So a little bit further. So there's a way to manage state that doesn't trigger a re-render because there are uses for that. That's a use ref hook. There's a way for passing values all throughout the React tree, the component tree, because sometimes it's really inconvenient or not. You can have a nicer API for your reusable components by putting some state in a kind of a shared pool of state, which we call a context. And then any component from that provider component on down can access that value of state without having to explicitly pass it as arguments to these components, which we call prompts. And so that's our use context hook. And there are various others for just facilitating those primary two use cases for hooks.

And then the thing that made Hooks such a game changer for the React community, was before we had to do all of this sort of thing within the different lifecycle hooks, not to be confused with React Hooks. But that's what we called them, was lifecycle hooks of the component. So we could run some code when the component initially mounted. We could run some code when there was an update to our component. And then we could run some code when it was un-mounted. And those were in three distinct methods on our class. And so you would often have multiple, or each one of those that was specific to a specific use case, or what you might call a concern. And so you'd have a component that could potentially have multiple concerns in it. And then the code for each one of those concerns was kind of spread throughout those methods.

The nice thing about – Like the biggest game changer about Hooks is that you – Because everything now is, instead of those class methods, you have these Hooks within the function body. You can co-locate all of the code for the concern into one area of the function body. And then if you decide, "Hey, I want to do the same thing in this other component." Then it's literally just JavaScript that you can reuse just like you do with any other JavaScript and you make what just a function, and then call that function into those two places. And we call that a custom hook. So it's a function that uses other hooks is what we call a custom hook. And those are typically prefixed with the word use just so that we can get some of the additional static code checks for us mostly from ESLint. And then also the convention can be useful for developers reading the code. So I realized that was certainly much more than a single sentence. But hopefully that gave people a pretty good overview of what React components look like these days.

[00:21:32] KP: Yeah, it's a great walkthrough. Thank you for that. Using a lot of those Hooks and some of the other tools you teach in the course, it winds up with this workshop phase where people are going to create their own Epic React app. I don't know if the students who go through it share their projects with you or things like that. But I'm curious if you've seen any good examples of fun things people have built.

[00:21:53] KD: Oh, sure. Yeah, so build an Epic React App. We build a bookshelf together. So most of the workshops before this one, the way they work is you just have a simple repo on your computer. And we have exercise files. In each one of those exercise files, there's an instruction document that you can read to get some background information to help you kind of get some context around what we're doing for the exercise. And then within the exercise file itself, there are code comments and emoji that are kind of there to help guide people through the workshop.

And so that's why if you go to my website, kentcdodds.com, you'll see koala is my company mascot, because Cody the Koala is the emoji that guides people through these exercises. With building Epic React App, that workshop works a little bit differently, because we're actually trying to iteratively build a workshop together. And so we have branches that you'll check out. And I have some scripts to help people get to exactly where they need to be. And then you still have exercise files, but they're kind of spread throughout the app to make it like an actual app experience where you're working in multiple files to accomplish a single task sometimes.

And so with that, people have taken that bookshelf app and added features to it. It's been fun to see what people will add to that workshop app. And then yeah, occasionally people will say, “Hey, I built this really cool –” I can't think of examples right now. But there definitely have been – People will share apps that they've built either for work and they'll say like, “I couldn't have done this without Epic React.” Or just like a toy app that they threw together for fun. I think somebody made a Clue game, like the game Clue where you have your scorecards or whatever. Somebody built that sort of thing to make it easier to keep score without killing trees, I guess.

So, yeah, I always appreciate when people share the cool things that they build based on what they've learned. And that's really the ultimate goal for me. My company mission is to make the world a better place by teaching people how to build quality software. And so I act under the assumption that most people, when they learn what I teach them, are going to try and make the world better with the software that they build. And so I want to help them do whatever it is they're doing better by teaching them how to build that software in a quality way.

[00:24:13] KP: One of the topics you cover was actually something I'm unfamiliar with. So maybe I need to go through that module, and it's React Suspense. Can you talk a little bit about what this is?

[00:24:23] KD: Yes. So for several years, I think it was 2017 at JS Iceland when Dan Abramoff introduced the idea of React Suspense and concurrent mode. I think at that time, they were still calling an Async React. But it's maybe a little bit too deep to go too far into it for this podcast episode. But basically, this feature allows React to render your React components. But if the user does some sort of interaction while React is doing that, it allows React to stop doing what it's doing. Because what action the user is performing might make that work unnecessary. Or even if it is still necessary, we don't want to block the user's interaction because JavaScript is single-threaded. And so it's a performance optimization.

But also what falls out of that optimization is the ability to optimize ever and pending states, because the abstractions that they had to build for this performance optimization kind of lends itself well for doing what we call suspending rendering of a component. And so what's interesting here, though, is in the last several years that they've been working on this very enormous feature, there have been a lot of changes and the way that this is supposed to work.

So the React Suspense workshop shows you what the world was like at that snapshot in time. But things have changed from that workshop, not in a hugely significant way, but a relatively significant way. And well, and the workshop also touches on things like avoiding waterfalls of – Like, often, we like to co locate our data requirements with a component that requires them. It just makes things easier to maintain that way. And so the problem with that is you don't get to start requiring that data or requesting that data until the component has rendered, which could be too late. Like it just makes things take a little bit longer. And so I teach a lot about how to implement render as you fetch. So you start the fetch as you start the rendering. And so that just speeds things up a little bit.

And so like there's some non-suspense specific features, some concepts that I teach in that workshop. But yeah, there have been a number of changes with Suspense since I made that workshop. And it's still a bit of a moving target. And I was really excited about Suspense until Remix came out and made it so all the problems that I needed Suspense to solve weren't problems anymore. So Remix is this framework by Ryan and Michael, who built the React Router. That is just a phenomenal React framework for building awesome applications, like my website, kentcdodds.com. And yeah, it eliminates a lot of the – Primarily the data, like loading data, showing error states that we get and really want from Suspense will still benefit from the concurrent mode features. But most of those you don't need to opt into. Those will just kind of happen automatically. So I'm not sure what the future of that suspense workshop will be. I'll probably just continue it in favor of teaching people how to build Remix apps, where lots of those data requirement features that Suspense kind of helps us with aren't actually needed.

[00:27:34] KP: Can you talk about Remix within the context of your own site? How do you apply it there?

[00:27:40] KD: Yeah, Remix is pretty much pervasive across the entire site. So my site is a simple Express app, which I deploy on a service called fly.io, in a Docker container. So whatever Docker container I stick up on there. And I can deploy it to multiple regions all throughout the world. And so I have a node server that's running – I think they've got something like 30 regions that you can deploy to. I'm only deploying to six based on my user analytics. That's all that I really need. And Remix, outside of that Express app, which is like, I don't know, maybe 200 line index JS file. Remix is a middleware that I stick in there, which handles the vast majority of my requests. And so then Remix has a server aspect to it for server rendering, as well as making data requests and that sort of thing. And then they have the client side aspect to it for routing. And there are so many things that I could talk. I don't know how long the show is supposed to be. But we're already at like a half hour. So I'm not sure how deep you want me to go into Remix. But it really is the foundation for my entire app. It's the reason that my app is really quite fast despite the fact that every single user has a unique experience on the website. So I can't do like just CDN caching or anything. Every single page is unique for every single user. And so

Remix helps to enable that being just extremely fast and also a really maintainable project for me. So I can go a lot deeper into lots of different areas of this if we've got time for it. But I'll just go and stop there.

[00:29:20] KP: For sure. Definitely. Well, yeah, I'd love to unpack a number of things. First and foremost, what about the user experience changes from visitor to visitor?

[00:29:26] KD: Yeah. Oh, and actually, I just remembered. There's one thing I wanted to make sure to make clear. And that is, in my app, I'm using Remix as an express middleware. But one of the really cool features of Remix is its ability to deploy to really anywhere. There's Cloudflare worker support. So you can even do like computation on the edge on a CDN, which is very cool. Eventually, they'll support service workers as well. And so you'll be able to do 100% client-side offline-first application with Remix as well. And then like Vercel, and Azure, and all of these platforms, you can use Remix with. So it's not like Express specific.

So yeah, anyway, on my site, the things that are unique for each user, of course, there's the user avatar in the top. And then every page has at the bottom some suggested content, because my site is primarily a blog. That's where most of the traffic goes. I also have two podcasts on there. One is the Call Kent podcast, where you can record a question or discussion topic right in your browser. Then I get notified of your recording. And I can go at my convenience to go and listen to it, record a response. And then I submit that. And that goes back to my server. I use FFmpeg to stitch them all together with really cool bumpers and everything and put them up on to a podcast feed. So there's like a really strong app aspect to my site as well.

And then I also have another podcast that I do like the traditional recording ahead of time with guests. And so there's that content. And then we've got workshops, and talks, and all sorts of things. So primarily, it's the blog content that I'm recommending to people at the bottom of each page. And that is randomly like – So there are three suggestions, and one of them is just randomly selected. Another is one of the six most popular. And then the last one is related to whatever content you're looking at. And none of these will be content that you've read in the last, I think, six months or so. And then of course, as you're reading a blog post, I keep track of how long you've been on the page. And so as long as you've been on the page for about 60% of the amount of time that I estimated should take to read the post, and you scroll down to the bottom, I mark that as a read. And so I never show you content that you've already read recently.

And then in addition to that, when you sign up for a user account, you can select a team. So it's like kind of like Pokemon GO where you have the red, yellow, and blue. And so you choose a team. And then relative to how many team members you have, your read of each blog post will count toward the ranking of your team on that blog post and on the entire blog as a whole. And so then based on which team is the leader for the particular blog post, all of the links and highlights and everything will be that team's color. And then there's a ranking chart on each blog post as well to kind of gamify it and make it kind of fun. It's been a lot of fun that way.

So in addition to just your recommendations, every time you go to the blog, we are calculating this team ranking. And we're going to show that team ranking to you as well. So there are a lot of really interesting and fun things that I can do. And I only did them because it was so easy to do with Remix. With my blog before, with Gatsby, you certainly can't do anything user-specific without showing spinners. And that's what I did on my workshops page, I would show a spinner while I determined whether or not there were any available tickets for my workshops. And I didn't have any user accounts because then you'd have to show a spinner for the user's avatar. And like Gatsby is the leader in what's called SSG, which stands for static site generation. But I lovingly refer to it as spinner site generation. Because once you try to do anything that requires some up to date data where you're not having to rebuild the entire application or part of the application to update that data, anytime you have to do any of that – So as soon as you have a user account, basically, you'll have to show a spinner. And I don't want to make that tradeoff. I just feel like that's not a good user experience. So I was really glad to move from Gatsby.

But next is also a good one for server side rendering. The thing that I really love about remix – Well, there are lots of things I love about Remix. And I'll stop talking here in just a second so you can ask follow-ups. But Remix just makes all these interesting and fun features that I was thinking about possible and easily done in a way that I feel is maintainable in the long term, which I did not feel from other frameworks that I've used.

[00:34:07] KP: So getting rid of spinners is kind of interesting. My experience in React is I'll maybe have a component load. Or I guess it's not a component anymore. I'll have my object load. It'll have a use state with some initial value. Maybe I'll kick off a hook to go call a REST endpoint to get the value. And before it comes back, I want to show a spinner. How do I get rid of that?

[00:34:30] KD: Yeah, so the way that you get rid of that is it's a little bit different depending on whether you're doing a document request or a script request. So a document request, you're coming to the page for the first time. That's going to be server rendered. There won't be any spinners anywhere, because you're creating all the HTML on the server side and just sending it all up as a response or streaming it. And so Remix, each of your route modules has the component that needs to be rendered and it also has an export that runs only server-side. It's called loader. And that is where you interact with any API's you want to go get your data, whether it be the transistor API for my podcasts, or the Tito API for my workshops, or even my own Postgres database with Prisma. And so I can go and get all the data that I need in the loader, and it's all running server-side. And then Remix takes care of getting that data into my component when it needs to render. And so on the server render, Remix will just say, "Okay, here are all of the route modules that match this route. And I will go and fire off or call all of those loaders in parallel so it runs really fast. And then we get that data back. And then I'll pass those on to the default export of each one of these routes. And there's nested routing. And so you have the root route. And then you have /workshops, /slug or whatever it is. And so each one of those will get rendered. And so that what's happens on the document request.

Now, once the app is hydrated, we're now doing client-side routing. Then the user clicks on a link. And so the URL actually gets updated. But before the router updates what component is showing, it actually makes a request to call off the loaders for the next route that's going to match and also does that in parallel. And once that data comes back, then it renders the components for the next page. And not just data, but also the code. So it's doing automatic code splitting and any CSS or anything else that I need any other assets I need for that page. So all of that happens before the router actually updates what's been rendered onto the screen.

And so for that reason, I never really have to think about pending states. I can. I can opt into those so I can show the user a nice pending state. If maybe I know that this page is always taking a long time to load its data, then I'd have a hook that Remix gives me to know what the next page is going to be. I could if, I wanted to, I could actually go with the old way of requesting it within the React component and show a spinner if that's the way I want to build it. But then to take things even further, Remix, you won't often find loading states for my app if you're using it like a regular user would. Now if you're like clicking around really fast and whatever, you probably could uncover a loading state.

But as you're hovering over the different links to the different places, or eventually Remix will probably add some additional smarts of knowing when you intend on clicking a link. So you won't even have to hover over it. But as you hover over it, or focus on it, Remix actually pre-fetches all of the assets and the data for that page. And so if you hover over something for a brief moment, then all of that data will actually get saved into the browser cache. So that when you actually click on it and Remix says, "Oh, okay, let me go get that data." It'll be right in the browser cache. And it's like milliseconds to get that out of the browser cache. And so there's no perceptible lag for navigating to the next page.

And for me, as a developer, I also don't need to think about pending states in every single component that I'm working with. I can. And what I do is I have like a global spinner that'll just show up after like 300 milliseconds if the request takes – Or the transition takes a while. But for most of the time, I'm not thinking about is my data here yet? Because it always is. And that is one of the things that just lifts an enormous maintenance burden off.

The other thing is that Remix route modules allow you to handle error states in a really declarative way. So your default export can always assume that the data is there, and there was no error in loading it. And then you have another export you can have on your route module that says, "Here's what I want you to show if there is an error." And so thinking about this is completely separate. And to take it even further, because Remix supports nested routing, your error state can be rendered in the context of the rest of the app. And so for the Call Kent podcast, each one of those rows that you have for an episode will just expand kind of like an accordion and show you the nested route, which is just the iframe for the transistor player, as well as the description and everything for the episode. So if there's an error in loading that episode, then I can have an error boundary there that I export from that module that says, "Hey, if this doesn't show up, then just say there was some problem loading this episode." And so the user can click on that. They'll see that error message. And the rest of the app is still working just fine. It's just that little part of the app that isn't working. And that also works with server rendering, which is actually a unique piece to Remix that they had to do a little bit of extra work, because error boundaries are a feature of React, but they don't work on the server. And so Remix team made it so that I can have this really nice declarative error handling API without having to worry about whether it'll work on the server. So these are just a couple of the things that make a Remix app really nice and maintainable and also just outrageously fast for the users. And a simpler website would be able to be way faster as well. Like if you don't have users, like you can increase the cache headers on your data and page requests so that it caches for an hour. Or then you could even better put a CDN in front of it and have that CDN cache for basically forever. And then you just purge whenever you have a content change or something.

So one of the things I love about Remix is how well it scales up and down based on the level of complexity of your app. And I don't feel like there's a correlated complexity metric there either. I feel like my app is – I've worked on a lot of apps and some apps that are shipped to millions of people and built by dozens of engineers. I've never felt so happy with the maintainability aspects of an app that I built. And so yeah, anyway, I've got so many other things I could tell you about Remix, but I talked too long. So I'll stop.

[00:40:55] KP: No worries. All really informative. I enjoyed poking around your site. I don't think we even got to cover all the bells and whistles that are there. It's great technology demonstration for sure. I'm curious if you've gotten any other feedback about it?

[00:41:08] KD: Yeah, so I've got this blog post titled How I Built a Modern Website in 2021, where I go into pretty good detail on the how I built a website in 2021. The blog post itself is 32 minutes long to read is the estimate there. So it is pretty lengthy. And I talk in-depth about a bunch of the technologies that I use, Remix included. And that blog post ended up on Hacker News, of course, as things do on Sunday. And I got a lot of negative feedback from people saying that it was highly over-engineered. And that is not surprising actually. In fact, I kind of like that, because it speaks to the fact that this is a real thing. Like there was a lot of serious engineering that went behind this. And the fact that they are claiming it's over-engineer just illustrates to me that they have no idea what this thing is capable of. Because they're saying things like – Well, of course, I didn't read the comments. You never read the comments on Hacker News. But I had other people tell me about the nature of the comments. And a lot of people, I guess, were saying, "Oh, you could do this really easily with WordPress, or Next.js or something." And there's no way you could do what I'm doing with WordPress.

Like for one thing, because every single page is different for every single user – Well, first, I guess is you can't deploy WordPress to multiple regions throughout the world. And so the solution to that is, "Okay, well, let's put a CDN in front of it. That is deployed throughout the world. And now it's super-fast." Yes, that's true. And it doesn't hit your origin server. That's awesome. Except you can't do that with my site because every page is different for every user. Okay, so now you're stuck.

So the fact is that, yes, you could build a simpler site with simpler tools. I wouldn't say it's over-engineered, but it is engineered. There're a lot of advanced things that I can do with this site. But my site, it's not one of those simpler types of sites. It's not just a blogfolio. But it's actually a legitimate app. And part of the reason why I wanted to do this is because as a teacher, when I first jumped into teaching full time about almost three or four years ago, I knew or I was concerned that I would lose touch with what it's like to build a real app with a team. And that's what I'm telling people, teaching people how to do. I'm trying to teach you how to build a real app with a team. And it's hard to do that if you haven't done it for a couple of years. And so I always figured, "Okay, well, I'll do this for a few years so I can focus on teaching and not have to take so much time doing other things. And then maybe I'll go back and build something." Well, I've done that now. I did have a team helped me with this. I had a UI dev who helped implement the designs that we had a designer make. And we had an illustrator, and a project manager. And like the whole thing. We had video editors, audio, people, everything. And so I have worked with a team. This codebase is approaching 30,000 lines of code. So it's not your blogfolio. It is a different thing entirely and has been extremely validating to all of the ideas that I've been teaching people for the last three years or so.

And now I'm super excited to teach people even more because things have changed in the last few years. And so, in particular, Remix is only a year old now. And it was in a developer preview. Just recently, they announced that they're going open source. They're going MIT. They're going to be free. Up until now, it's been a paid subscription model. And so I'm really looking forward to being able to teach people how to build better apps with Remix. And I've got workshops already scheduled that people listening to this could go check out my workshops page if they're interested in learning how to build apps that are both really great for the user and also amazing to develop with.

So yeah, you certainly couldn't do this with WordPress. Like I evaluated a lot of technologies when I was deciding, “Should I go with Postgres? Or should I try something with Firebase? Or should I use Supaass, or FaunaDB, or like it should I use MongoDB, or like where am I going to host the app and where am I going to host the database? And all of all of these decisions? Should I roll my own authentication,” which I did. And I have good reasons for each one of these decisions that I made. And I outlined them pretty well in this blog post. But of course, Hacker News doesn't read the blog posts. They just comment on the title and then they feed off of each other. But I feel like I've built the type of app that people are building at their companies. And I'm excited to teach people how I did it so that they can build better apps themselves.

[00:45:33] KP: Very cool. Well, remind us where the website is so listeners can check it out for themselves.

[00:45:37] KD: Yeah, kentcdodds.com.

[00:45:39] KP: Awesome. Well, Kent, thank you so much for taking the time to come on Software Engineering Daily.

[00:45:43] KD: Thank you. It's just been a pleasure to talk about all this stuff with you.

[END]