EPISODE 1494

[INTRODUCTION]

[00:00:00] ANNOUNCER: This episode is hosted by Alex DeBrie. Alex is the author of the DynamoDB book *The Comprehensive Guide to Data Modeling with DynamoDB*, as well as *The DynamoDB Guide: A Free Guided Introduction to DynamoDB*. He runs a consulting company where he assists clients with DynamoDB data modeling, serverless architectures and general AWS usage. You can find more of his work at alexdebrie.com.

Remix is a full stack web framework that lets you focus on the user interface and work back through web fundamentals to deliver a fast, slick and resilient user experience that deploys to any Node.js server and even non-Node.js environments at the edge, like Cloudflare workers. In this episode, we interviewed Ryan Florence, Co-Founder at remix software.

[INTERVIEW]

[00:00:57] AD: Ryan Florence, welcome to Software Engineering Daily.

[00:01:00] RF: Hey, thanks.

[00:01:01] AD: You're kind of well-known in the in the React community. You've been on Software Engineering Daily before talking about React a couple years ago. But I'd say more recently, what you've been working on is a pretty interesting new JavaScript framework called Remix. And that's what I'd like to talk about today. Maybe let's just start off there. What is Remix? What can you tell me about remix?

[00:01:18] RF: Oh, geez, I have been spending two years trying to figure out how to succinctly answer what is Remix? I think a term that we've come up with that I like the most is center stack. Remix is a center stack web framework for any stack developers. I've been doing software development, or web development specifically, for a long time. And that always seems to be the center of the stack where the frontend meets the backend that have really struggled.

And Remix is an attempt to solve that really well so that you feel like – It feels like writing a backend app and a frontend app. It's all the same thing.

[00:01:51] AD: Yeah, absolutely. I love that. I heard Kent say that yesterday, the center stack. And I hadn't heard down before. What is the big problem that we see, like, in these two separate worlds? What is the center stack solving here?

[00:02:02] RF: Another way we describe Remix is a browser emulator. Like I said, I've been doing this for a long time. Kind of the golden age for me of web development is when I figured out the LAMP Stack. It was PHP, and Apache, MySQL. What the heck is the L for?

[00:02:18] AD: Was it Linux? What is Linux?

[00:02:19] RF: Oh, yeah, that's right. Linux. Part of our stack was the operating system. You don't even think about that anymore. And the web server, Apache, like, who knows what is – It's probably nginx behind everything we're using today. But anyway, yeah, MySQL database, and then PHP. Just kind of like imagine just a login form, right? Just plain old login form. You've got the email address. You got a password field on a little form. And then when you hit submit, that was all just plain HTML back then, right? It was just an HTML form. The input fields had names on them. And the button said button type submit, right? And then when you click that, you didn't have any React state. You didn't have any Vue state, or bindings, or anything to these input elements, right? You literally just wrote HTML. And you gave some things some names.

And then when you hit that submit button, like, what happened next? You didn't go and grab all this state from a bunch of JavaScript and then like make a fetch post with like the content type of application, JSON. And like, await that thing, and have an error state, and have a loading state. None of that stuff existed. You simply left the user click the submit button. And by magic, the email and the password showed up in your server-side code.

It's not magic. It's the browser. The browser knows how to serialize a form and then send it over to your service. That's the center of the stack. The user just interacted. And I want to get from the frontend to the backend. That's the center. We'll get over there. And then I want to do some

processing over here. And then I want to send something back over the network. The browser also did that part for you.

Once your server-side code was done, it would either return an HTML response, or it would send a location header to like redirect you to a different page, to the dashboard, like if you logged in successfully. Or if not, it would send back some errors in an HTML page. A form caused a browser navigation. The browser actually like moves from one location to the next. The very same as clicking on a link. That's the center of the stack is this, like, the user did something and we want to get over to the server, do some processing, and then we want to get back over to the frontend to keep on going. And the browser used to fill that space. And then in the last 5, 6, 7, 8 years, a bunch of JavaScript now fills that space. And most of us don't do a good job as the browser did. But – Sorry. I can talk about this forever.

One last thing I want to say on this point is it's not just about creating those boring experiences, though, right? Like that's a boring user experience. A lot of times people see Remix or they hear us talk about it and they think that that's where our creativity ends. We're like, “We're just recreating PHP.” No, that's the model we want. That's the simplicity we want. We don't want all this extra state. We just want to have a form, send to the server and then come back.

But we also have been chasing this dream of really interactive web apps. And to do that, you need a bunch of state in the browser. And so, that's where Remix fills in with the center of the stack, too. Is it's like we're going to give you the model of the old school PHP. And a lot of people say it reminds them of Rails. It reminds them of PHP. It reminds them of .NET. It reminds them of whatever that golden age of web development was for a lot of us when things felt simpler.

But then we give you all the tricks that you need in the browser state. You don't have to manage the state yourself. Remix does. But it gives you things like there's a navigation happening. Here's the form data that's being posted so you can build some optimistic UI. Or we're going over to this route. And so, you can put a spinner on the button. Or you can fade out the main part of the page. Or animate the errors in when they come back. So center stack is emulate what the browser used to do so your code can be simple. Then add in all the bits to do the really dynamic UI that we've been chasing for so many years.

[00:05:59] AD: Yeah. Absolutely. I love it. I'm more of like a backend-y type engineer. Like, some of this stuff is new to me. I just want to do like a quick comparison of just like, “Hey, here's some other things I know. I definitely follow along with the frontend world. But I'm not as familiar. Like, I just wanted to like is this in the same category? Do they interact well? How does that sort of work? If I hear about like frameworks like Next.js, Gatsby, maybe even Redwood, like is Remix in that same sort of world? Are they like different options you would choose there?

[00:06:27] RF: It is less similar to those frameworks than it is to stuff like Laravel and Rails. It's also not quite like those either. It's both. That's why we call it center stack, right? It's not a frontend framework. It's not a backend framework. It's not a full stack framework, because we don't do like mailers and delayed jobs. Like, sometimes people coming from the backend perspective look at Remix and they think like, “How can they see this as full stack? It doesn't even have a mailer built in. Or delay jobs. Or whatever. Or a database abstraction.” Like, we don't have any of that. The whole point of Remix is to cross the gap. And they just kind of drop you off and be like, “Alright. Yeah, do whatever you want at that point.”

The big difference with the frontend – That's the big difference with the backend framework. It's just not the question you asked. The big difference with the more front-end-focused frameworks like Next, and Gatsby, or redwood is these still have that network gap. They provide ways to get data into a component. I want to load data from the server to do a pre-render, an SSG pre0render, an ISR, incremental static regeneration, or just as any kind of like – Or some of them even let you do as a dynamic server render, too. But they've got API's to get data from your backend into components.

But as soon as you have a button, or like a form onsubmit, or something like that, it's still up to you to now cross the network back over to the server. And they'll have like API route abstractions to say, “Oh, yeah, stick your server code here.” But still form onsubmit, like, it's up to you where Remix emulates the browser, and it'll serialize the form. It'll post it. It'll give you all the pending states in between. Catch the errors. It does all of it.

[00:07:59] AD: Okay, sounds good. Okay, so that's good for those sort of frameworks. What about React, Vue, Preact? Like, does Remix take an opinion on those? Does it work with all those pretty well? Or is that a bad question?

[00:08:11] RF: Since the beginning, we've been very careful to not call it a React framework. We love React. If anyone listening knows anything about us, we've been doing React for a long time. We had a company called React Training that we ran for six or seven years. And it's still alive. If you need some React training, give us a call. We have other instructors, though, who do a great job.

But yeah, at the moment, it feels very much like a React web framework. But we specifically don't call it that, because we realized while we were building it – I don't know. I'm just making up percentages here. But it feels like only 10% of Remix is actually React code. And we've been doing some work actually, the last few weeks, or a couple of months, pulling out the really interesting parts of this conversation we've been having, how to cross the center of the stack. We've been pulling out those parts out of Remix to move them over into React router. And so, React router is something that Remix is built on top of. But millions of web apps use it just as a standalone, usually as a client-rendered app router with push state and all that kind of stuff.

We've been pulling out all the Remix stuff to put in React Router. But we actually did. Instead of putting it right into React Router, we put it into its own package called Remix router. And that has most of the guts of what makes Remix really special. And you can – We actually developed it against Vue. The developer who's been doing most of the work, Matt Brophy, he used to work at URBN on a lot of their ecom stuff. And he picked Vue for their teams to use. He's a Vue developer. As we abstracted that stuff out of Remix, he actually had little playgrounds of Vue code instead of React.

[00:09:43] AD: Nice. Yeah.

[00:09:45] RF: Yeah. We intend to have first-class support for Preact in Remix. The React package is – Our scope on NPM at remixrun/react. Stands to reason, we can have remixrun/anything. We're definitely planning on Preact. Whether we asked Matt to do the Vue version or

not, I know he's going to. I think I'm just going to own it and say we're going to have first-class support for Vue as well.

There're other ones too. There's Svelte. There's solid. There's Marko. There's Qwik. Some of those are exploring really new interesting territory, like Qwik, with their – What do they call it? Progressive hydration. And their – I think they call it Bundle Shredding. Some of these frontend rendering libraries, they rely really heavily on the compiler. And so, I'm not sure where some of those are going to fit into Remix. We're not trying to have a strong opinion of like, “Oh, no, we only want you to use React, and Preact and Vue.” It's just some of them are exploring really new territory that our heads just aren't even in. We're excited to see where they land on that kind of stuff maybe one day.

[00:10:49] AD: And Remix kind of has its own compiler, right? Or, like, would that be a tough thing to interact? Or am I wrong in thinking of how Remix’s compiler works?

[00:10:57] RF: Yeah. We're kind of buttheads about our compiler. And one of the things – For me personally, Remix feels like a little bit of a correction in frontend web development. We kind of went crazy with compilers and bundlers. They're great. They're super handy. You can do some really cool stuff. But there's also like you start importing CSS files. And you don't even know how CSS gets on the page. And before you know it, you've got 500 kilobytes of CSS just all over, right? Or SVG. Like, people import an SVG file as a component. You render a list of 300 items, and you have an edit button, and a delete button, and like an arrow icon on it, right? That's three for 100. 300 in lined SVG things. Where you could have just referenced like one SVG file, and then just pointed the IDs of everything inside of it, right. There's a lot of things with bundlers that make it really easy to screw up the web performance of your app.

And so, we keep our compiler pretty closed at the moment. It's not a long-term opinion. It's just like performance is one of the most important things for us. So, we're going to keep that close for a while. But we have actually started work on making the compiler pluggable. Remix, the backend is pluggable. You can host it anywhere through our adapters. We just talked about how the frontend, not pluggable yet. The backend is the only thing that we have pluggable. But the frontend will be pluggable very soon. We're working on that. And we've just begun exploring making the compiler pluggable. Because we're going to have opinions on what your compiler

should be able to put into your JavaScript and what it shouldn't. But we don't have the opinion that we want to put all of our opinions on you, right?

We begin work on – We're calling it BYOD. Bring your own compiler Nice. Hopefully, hopefully, that work pans out. And then we can keep on being buttheads about whatever it might be. But then people can bring in our compiler plugins for other things. But that's super early work. We'll see. I don't know. It might not pan out. It might not be possible.

I've also personally been exploring no compiler, whatsoever. And just you've got a server. Let the server do what a compiler would do. And it's pretty cool running on Deno and Bun, these kind of emerging JavaScript runtimes. Yeah, I'm actually a little bit more excited about that. But anyway, it's all a screwing around and experimenting at the moment.

[00:13:16] AD: Yeah. I love it. You'll have like sort of a different opinion on how apps should be built and things like that, and different approach to it. Are there any like specific conceptual hurdles you're seeing over and over with people, especially if they maybe came up in the last five, seven years or so and learned it one way? Are there specific conceptual hurdles that you see coming up over and over again?

[00:13:35] RF: Oh, yeah. Newer developers have no clue what a form does by default. Like, they've literally never written a form without saying onsubmit. And then immediately, event prevent default. What default thing is it preventing? Like, what would have happened next? So that's a big deal for younger developers. And it's actually, like, I don't fault them for that. Like I, I'm part of the problem. Like, React Router is huge. And we've been – That's how I've been doing it for the last five, six years. How the heck should anybody who's new know that there's a different way to do that? That was even in the education space. I was the one who teach them to do that.

Just kind of knowing what HTML can do, what browsers can do. And then even things like the form data API. Like, that's a big revelation to a lot of people. Like, “Oh, even outside of Remix, if I got six form fields on this UI, I don't need six pieces of state, or a reducer, or any of that kind of stuff. I can just onsubmit, prevent default. And then I can say new form data, e.target, or current

target. And the browser and the DOM will say, “Oh, okay. I'll look through this form. I'll find all your fields, their name, serialize and then give you a bunch of values in this one object.”

And in fact, you can pass that form data straight as the body of a fetch. You don't have to set a content type or anything. You can just say fetch this form. And then the browser will do the rest for you. That's kind of a meta goal of ours is teach people all the great capabilities that are built right into the DOM and their browsers and HTML so they can use it successfully outside of Remix as well.

[00:15:08] AD: Do you guys do some with cache control and things like that? How does – For someone who’s not familiar with that, how's that work in Remix?

[00:15:14] RF: This one, we probably talk about too much, or we just talked about it poorly, to where people think that it's like this is the strategy of Remix. When it's like, Remix just speaks HTTP. Cache control is an HTTP header. And clients should respect it. And so, it's not like the only way to get high performance out of a Remix site.

But for us, the underlying conversation here – And I don't want to spend too much time on this. Because some people have very strong feelings. And I don't want to offend any listeners of your podcast. SSG, or static site generation, that meant a different thing to me when I was younger. Using like Nanak. I don't know if you remember Nanak and Jekyll.

[00:15:54] AD: Yeah. I know Jekyll. Yeah.

[00:15:55] RF: There's another one that started with H. I can't remember what it was. Anyway. To me, static site generation means you –

[00:16:02] AD: Hugo. Was it Hugo?

[00:16:04] RF: Hugo. That's it. Yeah. SSG to me that you kick out a bunch of static HTML files, stick them on a CDN, and like call it a day. But for a lot of developers, as I've been talking to them, and accidentally arguing with them on Twitter, SSG, for a lot of people, just means there's

some amount of static generation. Maybe just the shell. But then the middle gets filled in with client-side fetches and things like that.

I've learned there's some dissonance when I'm trying to talk to people about SSG. And when they're talking about it, there's a different conversation that happens. And I'm like, “Okay. I'm done talking to people about this.”

Taking the definition that I used when I was a kid with the Ruby generation SSG frameworks, you just have a pretty much or at least 90%, 95% fully formed HTML document. And you can get that without a build process. You get a request from the server, or from a browser to your server. You stick a CDN in front. Your listeners probably are more familiar with this. But yeah, a lot of people don't know how HTTP caching works.

And then it goes to your origin server. Origin sends back some cache headers that say, “Hey, you can hang on to this thing for a week, or a day, or a minute, or a year, whatever it is.” And so, then the CDN says, “Okay, I'll hang on to that.” And then it sends it back to the browser.

And then the browser actually has a cache just like that, that respects the very same specification, the cache header. The browser can also say, “Oh, I'll hang on to this also for a week.” And now when the user asked for something, the browser might just give it right back to them. Or if the browser doesn't have, it'll go to the CDN, and then the CDN might give it right back to them. The CDN is where you upload your SSG stuff. And then if the CDN doesn't have it, then it goes to the origin server and comes back.

Our new doc site for React Router, we haven't shipped it yet. But even in beta, where it doesn't get a lot of traffic, we've got some cache control on it. And we get 85% cache hits right now. And that's not in production. When we flip it into production, I'm pretty sure we'll be in the 90s. And so, really, the conversation here is you can either build those HTML files up front and then upload them to the CDN. Or you can just let it fill up as people visit the website. And the popular pages will always get fast responses. The unpopular pages aren't going to waste any resources. Yeah, cache control is another thing that people are very used to SSG. And so, we're like, “Hey, try this. Maybe it'll work. And if it doesn't, then go back to your SSG thing. That's fine.”

[00:18:22] AD: Yep. I legitimately do not know that the browser itself respects SSG with like its own local – Or like cache control with its own local stuff. I figured it would, yeah, go to at least to the CDN. And so that's –

[00:18:34] RF: Yeah. For HTML documents, it doesn't. Unless like you're offline and you click the back button, then it'll be like, "Oh, I've seen this. I'll hang on to it." But if you're online, it treats documents differently and it'll still go back and ask for it. But it's kind of critical for just creating really fast user experiences, too, even with nonpublic data, right?

Like, this conversation we've been having generally is for public data. And you can't really do SSG for private data, right? You can't log into Twitter and like – Yeah. And Twitter's like, "I'm going to render every tweet for every user ever."

There's a category of apps where SSG works. And there's a category of apps where like it's just not even viable. And that's where a cache controller can come in handy. You can even do really cool things. Like, you can vary on a cookie. The browser has a cache built into it. And so, a lot of times in React apps, people will like build these big caches with Redux, or React Query, or Apollo, GraphQL. Like, we've got a lot of these libraries **[inaudible 00:19:28]** manually. The user clicks around. Like, you've got stuff cached locally in-memory. But the browser cache exists and has a specification on it.

You can actually return cache headers from your data endpoints and Remix. And then there's another header called vary. You can say vary on the cookie. And so, cookies have just a string in them, right? And so, if that string changes in the cookie, then the browser will stop using that cache and it'll ask for the new thing.

Any data that user – Even private data. Any private, like a to do list or something, right? A Trello board. You can give really long cache headers on those. As the user navigates around, you don't have to go back and ask for it. And then anytime they make a mutation – And this is a big deal in Remix. We actually have a mutation story. Anytime you make a mutation, you can say, "Hey, just change a cookie to like the current timestamp." And now you've expired the browser cache from your server. And now the browser's like, "Oh, I'm going back to this page. Oh, that's expired. Because the cookies different, I'm going to go ask for the new data."

Yeah, there's a lot of cool things that the browser can do that HTML has, that the DOM has, that HTTP has. All these things are standards. Their specifications. They're not like – We're not making up our own acronyms here, right? It seems like every framework, anytime they make an announcement, they have a new acronym. And we have a no new acronyms rule at Remix.

[00:20:43] AD: Except for BYOC, the bring your own compiler.

[00:20:45] RF: Well, that's our internal ticket. Yeah.

[00:20:47] AD: All right.

[00:20:50] RF: Yeah, man. HTTP, it's pretty cool.

[00:20:53] AD: It's a good one. Yeah. Are there any sort of types of sites or types of applications that you'd say, "Hey, Remix is the wrong fit for this?" Or does it fit pretty broadly across all things, content sites, and as well as dynamic apps and things like that?

[00:21:05] RF: We get that question a lot. I worked at a company called Instructure. And they build a thing called Canvas LMS. Have you ever heard of it? Are you familiar with it?

[00:21:13] AD: Is that like in college? Like, the Blackboard thing?

[00:21:15] RF: Yeah. Yeah, it's not Blackboard. It's Blackboard, but a little better.

[00:21:18] AD: Okay. Yes. Yeah.

[00:21:20] RF: Or hopefully a lot better. I haven't been there for like a decade now. That was a Rails app. And if you did rake routes – So, if you're familiar with Rails, that will list out in the CLI every single one of the URLs that your app responds to. Because the way you configured your routes, it wasn't always clear how many there were. You could just like say, "Resource user." And that would actually create like six routes. And so, you'd say rake routes, and it would kick out how many there were.

In our app, when I worked there, it was like two and a half thousand routes. It was enormous. We had a – I can remember the consulting firm. We had a big consulting firm, a Rails consulting shop come in and help us do some refactoring. I remember, we had an engineering meeting with everybody in them. And the guy gets up front and he's like, “You know, we've worked on a lot of Rails apps over the years, we've never seen anybody take it as far as you.”

[00:22:11] AD: It's impressive.

[00:22:13] RF: The point is, it was a very, very big Rails app. Lots of UI. Tons of features. It was enormous. And since it was a learning management system, it was this nice combination of both app-like experiences. So, you're the teacher. You need to create the modules, syllabus. It's got drag and drop to reorder things. It's got like inline editing. Some of the pages have client-side routing. When I was there, that's what we started doing a little bit more. That was in the backbone era.

But then there were also very public pages that felt more like a WordPress site or a content page, where you're just looking at what is this assignment as a student, right? Or we had an inbox. Literally, an email client in there. We had live chat. We had discussions, where like students would have to like go post discussions online. I don't know if that was a good thing to get our generation into doing, since the first rule of the Internet is don't read the comments. We had a discussion board. Like, Reddit, right?

Like, I always joke that we had probably like name a startup, name a tech product. And Canvas had a version of that. Yeah. That, for me personally – And Michael brought his own experience working at Twitter, working a company called Path. A lot of social media, internal teams and stuff like that.

I also had a job as a – This is a very long answer to your question. I also worked at an agency building brochure websites, but also internal things for big banks and stuff like that. Our experience is the whole web.

When we built Remix, we weren't thinking, “We want something that makes it really easy to make a developer blog. Process Markdown. Syntax highlight. Static generate. Throw it up on the web. And you'll need a little bit of interactivity. So, let's have some like little bits of partial hydration and just this button is fun.” We came at this thinking about the entire web, and we've got experience on the entire spectrum of the web.

So yeah, I think if Remix is not a good fit for your website, then I feel like we kind of failed. I think it's more about how complicated do you want to get with deployment? It's not what type of UI or what type of website. It's more like, what infrastructure are you familiar with and do you like using? And SSG is super easy for your developer blog that has four posts? And will never get more. You’ll get your job. You’ll never touch it again. And those are nice. You can just like drag and drop a zip file, right? Or like, whatever. Hook it up to your GitHub and it'll build and push it to Netlify, or Vercel, or whatever you're doing.

But yeah, Remix forces a server on you. It gets all of its performance benefits. It gets all of its developer experience benefits. Because we're able to like – Server code is usually so much easier to write than UI code where you got to deal state, and pending, and UI. There's just so much going on. And Remix fills in that gap being center stack. So yeah, it's more about how familiar are you with deploying a web server?

[00:25:11] AD: I love it. I like to hear about your prior experiences, because actually want to dive in a little bit more on that. You mentioned both React training, React Router before. And I think people probably know you from that. Like, that's definitely how I know you. Let's start with React training, right? I don't know if it's the biggest React training group out there. But it's definitely like the premium one, or at least the one I've heard of, and things like that. I guess, how does your experience with React training influence creating Remix or how you're going about with Remix?

[00:25:38] RF: It was all the blank stares when we would make a normal form of the group.

[00:25:42] AD: I can't handle this anymore.

[00:25:44] RF: We're like, “How come nobody knows what I'm doing here?” They didn't know what the Marquee Tech tag was? Or the blank?”. No. I'm just kidding. So how did React training influence Remix? Is that the question? Is that right?

[00:25:54] AD: Yeah. Yeah, either in conception or in how you all create it now and market it and work on content, too.

[00:26:01] RF: It's kind of related to where I was talking about the experience that Michael and I have both had on the web. And Kent C. Dodds is also a co-founder. And he's been in the education space. He worked at PayPal. Did a lot of internal stuff there, too. That's kind of where he's coming out of from as well. But yeah, React training, we really just started that as a way to, I guess, fund React Router. We both had our jobs. We built React Router. Started kind of taking off. And people started asking us to come do consulting for their teams, or show them how to use React Router.

I did some internal training at Instructure on React. We had adopted multiple frameworks, backbone. We tried out Ember for a while. And then when we decide to go to React, I wanted to be really, I guess, intentional with that migration. And so, I set up actual training with our teams. And I made a little repo. There was a little half day workshop with exercises, and lectures, and code samples in there. And I put it up on GitHub, and people started emailing me saying, “Hey, can I use this repo? Or can you come train our team, actually?” And I said, “No, no, no, no. no.” And then I was talking to Brian LaRue. LaRue?

[00:27:07] AD: Yeah, LaRue. Yeah.

[00:27:09] RF: Yeah. I just realized I don’t really say his name out loud that very often. And I was like, “Oh crap.”

[00:27:13] AD: It’s all the Twitter people. Yeah.

[00:27:14] RF: Yeah. I have respected him for over a decade. I love his take on things. He and I are probably like 80% aligned on the web. And so, it's that 20% where we're different, where I love when I'm like, “I think he's wrong. But he's making me think.” And I love that.

[00:27:31] AD: And he'll definitely let you know if he thinks you're wrong. I love that about him. He's great at pushing that.

[00:27:36] RF: Yeah. I was talking to him at a conference. And I was just telling him about people were asking me to come train them on React. And he's like, “Why do you say no?” And I'm like, “I don’t know.” And he's like, “You should say yes.” I was like, “Well, how much can you charge?” And then he tells me. And my jaw hit the floor. I quit my job immediately. Started saying yes. Michael and I were both doing router. And he had people asking him to come and do some consulting and stuff. And we're like, “Let's join up. And this would be a great way to fund React Router.” So that's what we did.

And then we started expanding the team. Hired some more instructors. And then Michael went full-time on our open source. He did it for like six or seven months. He didn't do any workshops. Just worked on React Router V6. We hired Chance, who also works at Remix. Chance the dev as what people probably know him by. Chance Strickland is his last name.

And we had Remix on the roadmap. It was like – Chance worked on ReachUI. Michael is working on React Router V6. And as soon as they were done with those two things, we were going to like, “Okay, now let's do this web framework that we wanted to do.” Because I think Next is a great product. It didn't fit what I thought it was going to give me. And I didn't use React Router. It didn't have nested routing. And I was like, “I cannot build a web UI without nested routes.” We were like, “Yeah, let's take a stab at it. Let's make a framework on top of React Router.”

Basically, we shouldn't have called it Remix. We should have called it React Router SSR. Because we get all this pushback from people like, “It's so new. It's so new.” It’s like, “We should have just called it React Router SSR.”

[00:28:59] AD: It’s been around. The guts of it have been around for a while. You're saying, it’s using like what percentage of React apps –

[00:29:04] RF: Yeah, it's like 70% of them. So yeah, it was on the roadmap. So yeah, we really just wanted how do we server render React Router? And then the pandemic hit? And then it was just Michael and I sit in there. And we're like, “What should we do?”

[00:29:16] AD: Yup. Yup.

[00:29:19] RF: I was like, “I'm going to go cry in my room a little longer. And then let's come back and let's ship React Router V6.” And so, we worked on that for a week. And then I was like, “What am I doing? It is the middle of a pandemic. I just lost my main source of income. And I'm working on open source?”

[00:29:37] AD: Got to change the plan here. Yeah.

[00:29:40] RF: Yeah.

[00:29:41] AD: So then, you start Remix. And I think it's so interesting. Like, it starts with a paid product. And I knew some people that paid for it and love the thing. And then you raise funding last year. Tell me about like, that thought process of like, “Hey, let's do a paid framework,” and then maybe switching gears there.

[00:29:56] RF: Yeah. Well, we needed a new source of income. I actually, I saw it on GitHub when we were about to ship and when we were like, “Let's work on V6.” They have that little used by metric. I've never noticed that before. I've like, I've never counted the stars or the NPM downloads of our stuff. I was I was blown away. Like, I really hadn't looked at that stuff in a long time. I had no clue we had so many stars. And I'd never seen that used by thing. And I saw it there and I was like, “Holy crap! It said like 2.8 million.” I'm like, “If we can get a percent of a percent of them to give us like 200 bucks a year, like –”

[00:30:33] AD: Yeah. Where is that? Yeah.

[00:30:34] RF: Yeah. And it’s just a fun little project, right? I believe in open source. I also kind of hate open source. It's hard on you.

[00:30:42] AD: Yeah, absolutely. Especially something that widely uses that. It has so many different needs and requirements. Like, that's a big maintenance burden.

[00:30:49] RF: Yeah. I was like, “This will be fun to just have a little have a little project that –” There's a project in Rails, in the Ruby community. It's a delayed jobs thing that you can pay for **[inaudible 00:30:58]**.

[00:30:57] AD: Yes. Mike Pelham is the guy, right?

[00:31:00] RF: Yes. Job. Jober.

[00:31:03] AD: I know what you're talking about.

[00:31:05] RF: Cool jobs. Jobly.

[00:31:07] AD: Oh man. Sidekick.

[00:31:09] RF: Sidekick. Yeah, that's what it is. Michael and I were – Michael showed me sidekick. And we're like, “Let's just do that.” React Router is the free thing. Remix is the premium thing. And we started doing it. And when we figured out the mutation story, when we did the forms and the action, where – Like, I was talking at the beginning of this podcast, where we kind of emulate the browser. And then we'll give you all the pending states, all the form that it's being posted, all this stuff that gives you literally everything you need to build the fanciest single page application experience with the super simple PHP Rails, asp.net code. When we figured that out, it's got revalidation built in. You're not like crawling back up to the top of the app to like update the sidebar. It just felt so good.

I honestly couldn't believe that like the two of us came up with this framework together. And we'd been getting sales. Like, we were ramen profitable as they say at that point after just a few months. We're paying our mortgages with the Remix revenue. It would have worked. But when we saw just how good it was, and we had all these VCs emailing us, Michael's like, “Let's just talk to them.” I was like, “Alright. Yeah.” And after talking to them, we're just thinking about just how much we'd love to Remix. And it felt weird. I don't know. I don't know if the word is

irresponsible. It felt weird to just keep it to this as this like weird little group of 3000 people using this, right?

And yeah, we felt like it solved some of the major problems of frontend web development right now. And so, we were like, “Let's do this. Let's open source it. And we'll figure out revenue later. But this thing needs to be open source. And let's get some money. Let's hire a team.” That was the plan though, too. Like, we're going to sell licenses. And then we're going to figure out another way to make money. And then we were going to like – Kind of like this like trail, right? Like, first you sell the thing, then you figure out the next thing to sell. And then you'll open source the thing behind that. That was always our plan. Remix was always going to be open source. It was just a matter of timing. And so, we took the money as just a way to like just accelerate that.

[00:33:07] AD: Yeah. And I don't know if you're sick of hearing this, but have you thought about business model type stuff? I mean, I saw you hawk in apparel on Twitter earlier today. Do you have thoughts on that? Or are you far enough off that’s like, “Hey, let's get some adoption. Let's build a community. And –”

[00:33:21] RF: Yeah. I mean, we have enough runway to not have to think too much about that. But we ran a profitable company through six or seven years. So, revenue and profit is just in my blood. And it's kind of annoying for me to like be working on a project that doesn't have that coming in, in some form or another. Like I said, I don't pay attention to GitHub stars. I pay attention to my Stripe payments. That's what I'm interested in.

Another joke that we say is we're just going to do conferences, because we had a wildly successful conference for a six-month old project. That was crazy. But no. And I'm deflecting. So anyway –

[00:33:55] AD: How big is the no?

[00:33:56] RF: We have several ideas. We have like three or four ideas on what we want to do. And it's probably not hosting. That's where people usually think, “Oh! You'll do the Vercel move and have your thing that brings you in?” But one of the things that we really love about Remix is

that you can deploy it anywhere. And there's so many cool new things showing up on the backend that we would hate to introduce a weird incentive for us to like, “Oh, we don't want to support Deno because we don't make money if you do that.” Probably won't be like direct hosting of Remix. But there's a lot of services we know that we can build on top of it. And those would have their own CDNs, have their own servers, like maybe images, or translations, or whatever, right? There's a lot of ways we can go.

[00:34:36] AD: Yeah. Cool. How big is the team now?

[00:34:39] RF: There's nine of us. So, all engineers and one designer. But he's also really great engineer, too. So yeah, too many engineers. We should probably start hiring some other roles.

[00:34:51] AD: Cool. Cool. All right. Just to close this out here, I have some questions just like on future work. Like, hey, is this ever going to get in remix? Or is it just like philosophically opposed to how Remix works? So, models, right? In the documentation I look at, it's like, hey, Remix is like the Vue in the controller in MVC. It doesn't do anything with models. It doesn't take a database preference or choice or anything like that. Is that something you think will happen forever? Like we ever get into models and have that? Or just be like, hey, bring your own models? Plug it in. However you want to do it?

[00:35:19] RF: Yeah, that's also another potential revenue stream. But no, we don't have direct plans for that. So, Hasura, Prisma. You're good. We still love you too. Sorry. Sometimes I know my jokes would be like interpreted as something.

[00:35:31] AD: Yeah. Start a Twitter storm here.

[00:35:32] RF: Yeah, exactly. No. Yeah, models. There're so many cool things out there, right? Prisma has is kind of the active record for Remix. If that's – If we want to have a Rails-like feel, there's no reason for us to come up with one of those. Prisma is great.

FaunaDB, super cool, distributed cloud storage, which fits our like edge deployment story really well, is there too. So yeah, there's so much innovation happening in that space that I can't imagine us wanting to get super involved there.

The hardest part of web development, in my experience, is that center stack. Like, how do you connect the UI to these backend pieces? There're so many good ways to do – I mean, that's right. You're the DynamoDB dev. I want you to be able to use DynamoDB with Remix. Like, I don't want anything that like pushes you to some other kinds of stories.

[00:36:24] AD: Yeah. One thing that's cool about Remix, right? It's doing all these fetches in parallel. So, you don't have this waterfall type thing. As it mentions in the doc, hey, that means like each route has to authenticate, authorize itself. Is that something that's always going to be true? Like, we just can't get around that sort of thing? Or will it be some way around that going forward?

[00:36:43] RF: Yeah, that's a good question. Yeah, we're working on, I guess, a middleware. I'm hesitant to use that word, because that brings a lot of baggage with what people think it'll mean. But with React Router just in the browser, it is a common pattern to have like a parent route, do some auth check in the browser, and not render its child routes if there's not a user, right?

And so, we kind of – Or is it Tony Stark, who says that we create our own demons? We kind of set this precedent in React Router, we're like, "Oh, a parent route can like block the rendering of the children." But if you're going to fetch all those in parent, the data for each one of those routes in parallel, like you can't block like that, or you're going to slow everything down.

We've got some designs for the ability for a child route to be blocked by some sort of hook on a parent route. Just for a sense of what it might look like, like a before loader. You can export a loader for each one of your routes. And we'll call those in parallel. If you export before loaders, we'll actually call those serially first and be like, "Okay, you want this route data? First, I'm going to call those.

The thing to recognize, though, is when you send your jot from like an SPA up to your API routes. And you fetch three or four API routes all with the same jot, it's getting validated in all four of those requests. That's the funny part about this question, because it comes up all the time. And it's like, "Well, look at your SPA. You just didn't write the backend. So, you don't see

that it's checking for the user. Now you're writing the backend. So, you see that it checks for the user." So, it's really not a big deal.

[00:38:12] AD: Yeah. Last question on Remix future features. When am I going to get my mailer? My delayed job? So I can switch over from Laravel?

[00:38:21] RF: Yeah. Use Laravel. It's got great backend features. Doesn't AWS have a mailer?

[00:38:27] AD: Yeah. I'm just joking about that from earlier about where's my mailer? But, yeah.

[00:38:30] RF: Yeah, yeah, yeah. But no, that is our philosophy. There's a lot of great services out there. There're no libraries for it, too. So, yeah.

[00:38:38] AD: Yeah, yeah. Good. I love this. I've been hearing a lot about Remix. And it was great to like hear what's going on there, some of the background. Anything else I should know? Anything you want to hit your plug or let people know how to find you? Anything like that?

[00:38:50] RF: I want to just talk about streaming real quick.

[00:38:51] AD: Yeah, let's do it.

[00:38:52] RF: If that's all right. This is a big feature that will be shipping really soon. We've kind of been just telling people like, you just got to make your backend fast. Then make your app fast. And people are like, "My backend is not fast." And we're like, "Yeah, we know. Go make it fast." "Just install Redis." or Elasticsearch or Elasticcache? Is that what it's called?

But that's the thing. Like, a backend engineer who's like got a lot of experience back there, they know, yeah, that's how you make a backend fast. It's like I sometimes joke that Remix is actually a lot easier for backend devs at the moment than frontend devs. Because most of the strategies that we recommend are backend strategies. And you want to make a combo box fast? This is like searching, hitting the server, put a full text index search on your Postgres table, right? Don't do some sort of weird trick with JavaScript. The database will make that fast.

Streaming is really cool because you don't always have control of your backend. And you can't always spend the time to make it fast. Streaming, to me, is a frontend trick. It's a rendering trick to kind of like hide, just like optimistic UI. Optimistic UI is this idea that like I clicked a button to delete something. I can hide that right away even though I haven't talked to the network. I haven't even told the server. And it's probably going to be correct. So, I can just hide it. It's these tricks to make an app feel fast. Streaming, to me, is one of these tricks.

And what's really cool about streaming is if you've got – In Remix, especially. In Remix, we're able to flag any piece of data, not just a route and a message route. Like, you don't just flag a whole route. But you can flag three pieces of data inside of a single route. Just think of like the three boxes in the UI. Maybe it's an ecommerce site, and it's the reviews and the comments. Maybe those reviews and comments, you don't even control that API. It's a third-party service. You can mark that as deferred. We're calling it the deferred API's. You say, “Hey, defer the comments. Defer the reviews. But don't defer the product data.”

And so, Remix is going to wait for the product data before it renders. Because we call that critical data. That determines your status code. Is it a 200? Is it a 404? You don't want to send a request if you don't even know you got the real thing, right?

But let's defer the things that are going to slow this thing down, like the comments in the reviews. And Remix uses React 18 streaming, even without server components yet, if you're familiar with what they're working on over there. They've got a handful of API's that all work together. And server components is the one that's not shipped yet. We don't use that one. We don't ship unstable API's. But it'll stream in with React, the shell of the app, and the product. And then the reviews and the comments will fall back to a spinner.

But what's so cool about this is the back end kicked off the fetch for all three of those pieces of data, probably four. You're probably getting the cart and the user in the top-level route, too. So, you got six pieces of data. I think I just added one more. Five pieces of data. Yeah. That are all kicked off on the server in parallel instead of waiting for JavaScript. Most web apps, JavaScript lands in the browser, and then we start fetching data. All of them started on the server. We stream down what we know. And then as those pieces land from the server, they pop in. So, you want a really good skeleton UI.

But what's really significant about this is CSS assets, and fonts, those are render blocking. Even if you have everything, if you have all your HTML, you have all your JavaScript, you've completely hydrated everything. If you're still downloading CSS or a font, the user still gets a white screen. No matter what other tricks you're doing, you want to start downloading CSS, and fonts, and JavaScript as soon as you can.

Streaming is really cool, because it allows us to kick off those five data requests on the server, send a little bit down to the browser. And now, the browser, since it has the shell of the app, it can say, “Okay, what are all the assets that I need?” And now the browser is going to start downloading as well. You've got the backend fetching data. And you've got the frontend fetching resources. And it's all in parallel.

It has enormous effects on perceived performance, even if your API is a little bit slow. Because everything happens at once. Web performance is mostly about the order in which you do things? And it's all about parallelizing. So yeah, we're hopefully shipping that in the next month. It's looking really good. Oh, my gosh. I'm so pumped about it.

[00:43:16] AD: That's cool. I love it.

[00:43:16] RF: But you should probably just make your backend fast.

[00:43:18] AD: Yeah. Do that first, kids. Put speckles of Redis on it, and then use deferred when –

[00:43:23] RF: Yeah, sprinkles of Redis. Yeah, that's what I'm most excited about. And I want people to know we're about to ship. It’s going to be pretty cool.

[00:43:29] AD: Cool. Awesome. We'll be looking out for that. Where can people find Remix? Was that remix.run is the website?

[00:43:34] AD: Remix.run is the website. Remix_run on Twitter and GitHub. Actually, GitHub is -run. We got a branding problem. But you can find them all at our website. We've got all the links up top. And then I'm just RyanFlorence on Twitter. All one word.

[00:43:49] AD: Cool. Ryan Florence, thanks for coming on Software Engineering Daily today.

[00:43:52] RF: Yeah. Thanks, Alex.

[END]