EPISODE 367

[INTRODUCTION]

[0:00:00.8] JM: Airbnb is a company that is driven by design. New user interfaces are dreamed up by designers and implemented for web, iOS, and android. This implementation process takes a lot of resources but it used to take even more resources before the company started using React Native Native. React Native Native allows Airbnb to reuse components effectively. React Native works by presenting a consistent model for the user interface regardless of the underlying platform and React Native emits a log of changes to that user interface so that the underlying platform can translate those changes into platform-specific code.

Leland Richardson is an engineer at Airbnb. In today's episode, he explains how Airbnb uses React Native, how React Native works and the future of the platform. This was a great episode, it's a deep dive into React Native and I think you'll like it.

If you're curious about finding specific episodes of Software Engineering Daily, I recommend checking out our new topic feeds in iTunes or wherever you find your podcast. We have sorted all 500 of our old episodes into categories, like business, or blockchain, or machine learning. We've got a greatest hits feed that I endorse strongly. It's got all of the curated best episodes. If you have trouble finding the episodes to listen to, like many people, that's why we made these topic feeds. Whatever specific area of software you're curious about we have a feed for you and you can check the show notes for more details. I would love feedback on this if you have more ideas for what we can do to improve. Send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[0:02:01.0] JM: To build the kinds of things developers want to build today, they need better tools, super tools like a database that will grow as your business grows and is easy to manage. That's why Amazon Web Services built Amazon Aurora; a relational database engine that's compatible with MySQL or PostgreSQL and provides up to five times the performance of standard MySQL on the same hardware.

Amazon Aurora from AWS can scale up to millions of transactions per minute. Automatically grow your storage up to 64 terabytes if need be and replicate six copies of your data to three different availability zones. Amazon Aurora tolerates failures and even automatically fixes them and continually backs up your data to Amazon S3 and Amazon RDS fully manages it all so you don't have to.

If you're already using Amazon RDS for MySQL, you can migrate Amazon Aurora with just a few clicks. So, what you're getting here is up to five times better performance than MySQL with the security, availability, and reliability of the commercial database all at a 10th of the cost. No upfront charges, no commitments, and you only pay for what you use. Check out Aurora.aws and start imagining what you can build with Amazon Aurora from AWS. That's aurora.aws, A-U-R-O-R-A.A-W-S.

[INTERVIEW]

[0:03:40.4] JM: Leland Richardson is a software engineer at Airbnb. Leland, welcome to Software Engineering Daily.

[0:03:44.8] LR: Thanks for having me.

[0:03:47.0] AS: Today we’re going to talk about React Native and cross-platform development and some of the technologies that you've been building at Airbnb. Let’s start off with the canonical question of cross-platform development, which is — Airbnb has to be written three times for web and android and iOS or at least it did prior to starting to work with React Native. How did this process work before React Native? For the companies that are out there that are still writing web and android and iOS apps to maintain cross-platform capabilities, what are they doing and what breaks down in that process?

[0:04:31.0] LR: Most companies kind of have those three targets. You have web, android and iOS. Some might have additional targets, like a desktop app or Windows phone, things like that. What you find is a lot of times when you're writing these clients for different targets that you’re actually writing more or less the same code every time, but you can’t share that code because you're using different technologies differently and things like that. Airbnb, like pretty much every other company, has been in that position and still for the most part is. React Native is starting to change that a little bit.

Just to be clear, Airbnb is not 100% written in React Native. It's still a pretty small minority of the product that's written in it. Right now, we use React Native for just iOS and android cross-platform development and we're starting to explore using it for web as well.

[0:05:41.5] JM: What are the problems that arise when this engineering process is siloed by platform?

[0:05:50.1] LR: Yeah, that's a great question. I think a lot of people view it as just sort of the way things are and the way things ought to be. It's just kind of been that way for so long. We've had some other cross-platform technologies, but for the most part people have kind of come to terms with it. They usually just don't work all that well.

A lot of what can come up with that, until recently, Airbnb was — We kind of had these product teams that worked on specific product silos. You might have one team that works on payments, another team that works on search, things like that. The way that traditionally kind of worked out was those teams that the product engineers on those teams were either back-end engineers or front-end web engineers. Then we would have individual android and iOS teams.

You ended up having this really big communication problem where we didn't have enough iOS or android engineers to kind of spread them across the whole organization, but they still kind of needed to build the whole product. You ended up having a lot of sort of first-hand knowledge that the web engineer might have that would have to then get transferred to the iOS and android engineer and it resulted in, basically, a lot of features that weren't fully implemented on the native apps or at least some sort of lag in terms of when they would get implemented. You essentially have the whole organization kind of fighting for these limited native engineering resources.

[0:07:35.2] JM: When React Native Native first came out, it did not claim to solve the problem of siloing, but it did offer some improvements in communication and information sharing and cured some of the process of this cross-platform disaster. Overtime, it's gotten better. Explain how React Native changes the development process for cross-platform developers.

[0:08:05.7] LR: Yeah. When React Native first came out, the kind of tagline they chose was learn once, write anywhere, which is sort of a play on the kind of the Java, historical Java timeline which is like write once, run anywhere. The idea they were trying to kind of say like, "Hey, we're not trying to do that exactly. That what we're saying is we have this kind of paradigm, this way of writing UI and writing applications which is essentially the JavaScript and Reacts stack, if you."

Basically saying, "We can take that and can we write each application in that stack or in that paradigm even if we might have to write it twice or three times or whatever," that because we've only had to — Because we're using the same stack every time, we're able to share the same kind of engineer to do that. You can have the same person implementing it in all three areas and they might have all of the product context of how to build that application and things like that.

Initially, that was kind of the selling point and I think that that worked out really well. What's really interesting though is you have this effect where before you had — You maybe wrote it three times, but you had three different engineers writing it then or the people writing it for iOS weren't the same people writing it for android, weren't the same people writing it for web.

Once those start to become the same people, then all of a sudden the sort of pain of that duplication starts to feel all the more real. You start writing on web and you start writing it on iOS and android for React Native and then you realize just how similar those two applications are and you realize that there aren't — The number of kind of differences that you bake in, do to the platform, in a lot of products is actually incredibly small. It's just the technology that you're using that ends up making the code different.

Once you make the technology stack same, it starts to kind of eke at your inner engineer that doesn't want to repeat the code that you're writing all day. You kind of get this interesting effect where you're doing less work than you were before, but for that individual engineer it might actually feel more painful.

[0:10:52.7] JM: What you’re describing is when a company like Airbnb goes from writing the app on native iOS, native android, and React on the web or any JavaScript framework, really, to a place where they’re writing a React Native app that can basically be deployed to iOS and android and building a React app on web, you go from three platforms to two platforms. That's good, but the downside is it lets you identify just how inefficient things are.

[0:11:32.9] LR: Yeah, exactly.

[0:11:33.6] JM: Right. At this point, you would have React Native, React Native that suits iOS and android you you’ve got React and JavaScript on the web. What’s the difference between those two environments? Why can't I just port my code easily from one to the other?

[0:11:56.4] LR: Yeah, this is a problem that I’ve been kind of focused on and thinking a lot about lately, but the differences end up being really important but really subtle. You can kind of identify that the code you’re writing is effectively the same, like in the way that you care, but you can't just kind of copy and paste things over and rename things. It’s not quite the same. For application logic, maybe a lot of it is, but for UI logic you end up in some places where you don’t realize how you're depending on the platform.

Typically, when you write a React web application, you're kind of directly targeting HTML, and so your React components are going to be rendering directly. They’re going to be returning these HTML elements like div and span and H1 and H2 and things like that.

What’s interesting about React Native or about React is that you're not actually using the browser APIs when you're writing those components but you are targeting them. That div or span in your JavaScript code is actually just a string, but that string is eventually going to turn into a div on your actual website. In the React Native world, you don't work in div and span and H1s, things like that. They have a different set of primitives, like view and text and image and there are a couple more, but those are kind of the core UI primitives.

The UI primitives for React Native feel very similar to the ones on the web, but there are a couple of important differences. One of them is the layout algorithm. When you start writing divs and spans or views and texts, you're implicitly relying on the way those things get laid out very intimately and it's very important for building the actual UI that you want to look at later on. React Native early on shows to implement Flexbox; a subset of the full CSS spec that the web builds. You have access to the Flexbox layout algorithm on the web, but it's not the default algorithm.

React Native kind of works on this subset of CSS. If you want to write cross-platform code, you have to essentially build the equivalent of these React Native primitives that have the same constraints that the ones in React Native do. You then sort of constrain yourself to using the Flexbox layout algorithm which actually is a very nice layout algorithm. It's a little bit newer in the web spec, and so it's not used as often, but it covers a lot of things, like a lot of UI paradigms that used to be very hacky and hard to do are very easy to accomplish with Flexbox.

[SPONSOR MESSAGE]

[0:15:23.5] JM: Artificial intelligence is dramatically evolving the way that our world works, and to make AI easier and faster, we need new kinds of hardware and software, which is why Intel acquired Nervana Systems and its platform for deep learning.

Intel Nervana is hiring engineers to help develop a full stack for AI from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for an opening on the team. To learn more about the company, check out the interviews that I've conducted with its engineers. Those are also available at softwareengineeringdaily.com/intel. Come build the future with Intel Nervana. Go to softwareengineeringdaily.com/intel to apply now.

[INTERVIEW CONTINUED]

[0:16:19.5] JM: So, then what's the current status of bridging that gap between my React Native for iOS and android app and the React app on the web? Is the current status quo good enough to bridge that gap? Is this a problem that's going away?

[0:16:40.0] LR: I think it really — At the moment, I think it depends maybe on your use case and what sort of browser support you need and what your sort of constraints are. Right now, there's a library that's written by Nicholas Gallagher from Twitter called React Native Web, and it's essentially a set of the React Native APIs that React Native exports implemented for the web. Among those are what I sort of call the primitives and are sort of I think the most important ones, which are you view text image, style sheet, touchable and animated. We don't really have to know exactly what all of those do, but you can build most interfaces with just those 6 primitives.

React Native web does a great job of implementing these on the web and actually Twitter Lite, the new PWA that Twitter put out for mobile twitter.com is built on top of React Native Web. This is being used and sort of production scale in some places.

You should be very aware of like there are some performance implications to using these primitives relative to just using like div and span directly and that Nicholas and other people in the community, like me, are sort of currently working on trying to tighten that gap in terms of performance and kind of remove that as an issue.

[0:18:17.2] JM: Are there runtime performance issues?

[0:18:20.0] LR: Yes.

[0:18:20.9] JM: Okay.

[0:18:24.3] LR: Oh, sorry.

[0:18:24.5] JM: No. No. Go ahead. You want to talk about them a little bit?

[0:18:27.4] LR: Yeah. I was just going to say. I think that more and more, there's an importance of the ability to keep your JavaScript bundle small. The JavaScript parsing execution times of like old android phones is really bad.

So, I think with frameworks like this, especially where the typical use case for a React Native wet at the moment might be for writing actual mobile websites, and so it's pretty relevant to keep those times as fast as possible. I think that there is still a lot of improvement that can be made there, but I'm pretty excited for where it's headed.

[0:19:14.0] JM: To clarify what we're talking about here, I think we should go into a discussion of React Native APIs and what that actually means. There's all these different APIs that you can use in React Native and I know that you wrote a script to find the most frequently used APIs. I know you found thse seven APIs that you mentioned were the top ones and they accounted for something like 85% or something. I guess before we go there, we should about that even means. What is a React Native API?

[0:19:50.9] LR: Yeah. There's a JavaScript library called React which is one of the dependencies of React Native, and so React Native and React or not like siblings of one another. React Native is kind of an all-encompassing framework. Whereas React is just a JavaScript library that React Native itself depends on.

You can look at React Native as a set of APIs that you can use in JavaScript to target like native apps. The actual JavaScript that's running is run in a JavaScript virtual machine that's actually running on your device, like in the app. That JavaScript is then communicating with simple message passing across what we call the bridge in React Native, from the JavaScript's context into the native context. That communication layer kind of amounts to all of all of APIs that you would typically need to build a mobile app.

React Native itself has tried to kind of build these small APIs for every — Not every, but for a lot of the typical app use cases. Sometimes that's like a UI thing, like view and text are some of the UI components, but there's also things like switch and text box, scroll view, things like that. There's also non-UI components, and so there's a network information API. There is an internationalization API. There's a geo-location API. There's a network layer. There are things like that that aren't really React related, but they're just APIs that React Native exposes to you by default.

[0:21:56.9] JM: We should clarify here how React Native and JavaScript are running on, for example, iOS. People think of iOS typically as swift code or Objective-C code. For React Native, there is this bridge to getting to be able to interoperate with an iOS app, for example. We've done a lot of shows about this, so people can listen back if they want to hear about this discussion in more detail. Maybe could give an overview for — Before we go back to the React

Native APIs, in particular. How is React Native this thing that interacts with a traditionally swift or Objective-C app in the case of iOS?

[0:22:47.9] LR: Yeah, sure. On iOS, it's actually pretty easy. Apple actually includes in the operating system a JavaScript core API, and so when I say JavaScript core, what that is is a JavaScript virtual machine, a JavaScript runtime that Apple has made on top of WebKit. That's packaged with the operating system. If you're writing an app, you can instantiate an instance of JavaScript core and you can inject JavaScript's code into it and it will run. That's essentially entirely how React Native works. They've built what we call the bridge which is essentially just a message queue, a bidirectional message queue across this JavaScript core interface where you can injects objects into — Or messages into the JavaScript runtime and then the runtime can pass messages back to the native runtime. They've essentially created a very lightweight protocol to turn those messages into actual commands that resulted in UI and in things being drawn to the app and doing actual native things.

[0:24:15.0] JM: When I write a component for React Native, what's happening is that component is getting, I guess, processed by that JavaScript virtual machine and then its passing messages over that bridge to the native code, I guess?

[0:24:35.5] LR: Yes, this is actually I think one of the more innovative kind of pieces of the React JavaScript library that maybe people don't quite get if they're just casual React users. React is a way of building up your application as a set of components that are essentially functions that take in a set of props, like some data and then output a description of what the corresponding UI might look like provided that data.

It's important to understand that that literally is just a description, so it's not the actual UI. It's just turning this UI into serializable data and then the React library looks at that data. This data, a lot of people refer to it as the virtual DOM.

The React library itself looks at any state in time. It looks at this data and compares it to what it looked like earlier and finds the differences. Whenever there are differences, it turns that into a command. A command would be like a create view, or update view, or add child, remove child, move child, things like that. Those commands are essentially like a serialized list of instructions and that kind of works that way in React web for the browser, but it turns out that you can — These instructions are pretty much generalizable, and so you can instead take these instructions and pass them to an iOS application and have them create UI views and kind of instrument the view with those instructions instead of browser, like DOM elements.

[0:26:36.1] JM: Basically, an event log of changes to a UI and it's basically platform agnostic because you just have event log of changes that should be made, and whether those changes are being communicated to an android client or to an iOS client or to a web client or to a VR client, you have the same sorts of operations that you want to do and you leave it to the underlying platform to translate that event log into actual UI.

[0:27:16.3] LR: Exactly.

[0:27:18.7] JM: Okay. That settles for me the decoupling, because then, okay, we're just talking about like what are you wanting to do at the Reacts native layer to best create this event log of stuff that's going to turn into native UI components. We've got the seven fundamental APIs that we're going to build a user interface through. You got components, layout, interaction, animation, and branching. Wait, I'm sorry. No, those are — Sorry. Those are the —I should have said this more clearly. Okay. You've got the seven APIs and then you categorize them, you informally categorize them as APIs that fall into the category of either components or layout or interaction or animation or branching, these different like kind of adjectives that you want to classify these APIs as.

Maybe you could talk again about the seven APIs and why you classify them in these in these different ways and kind of how they fit into these different classification. I should emphasize, I think these are informal classifications that you've basically kind described them as.

[0:28:31.5] LR: Yeah. Maybe — I don't know if we've kind of explicitly said it, but the seven APIs we're talking about are the APIs that I've kind of dubbed as the primitives. Released to a library called Reacts Primitives that has these seven APIs. It's important to understand that this library called React Primitives is kind of just an interface. It's just a concept. There's actually — In that library, there is no implementation.

What I’ve tried to identify is that the real trick here is really just identifying what the valuable primitive interfaces are and what they actually represent, because React kind of does the hard part of really decoupling this entire application code from the underlying platform. Now that we have that decoupling, React Native is a really good example of these core APIs being implemented on two relatively dissimilar platforms; iOS and android, but allowing them to achieve like very very similar results. All we've done or all Nicholas Gallagher with React Native Web has done is created that same interface for the web as another platform.

I think of the web as just another platform. I think that there are many more. Some that we may not even have invented yet, but maybe invented in coming years that I think this could be an important concept for.

[0:30:21.6] JM: I think one way to illustrate the importance of the concept is that in computer science classes, if you take them in college — Actually, if you go to a coding boot camp too, you learn about these core data structures, like linked list, or array, or hash map. We've kind of like settled on these core primitives that we use to build application logic, and like those haven't really changed much in a pretty long time. Everybody uses hash maps. Everybody uses linked lists. Everybody talked about the trade-offs with these things, but we never really had a good way of talking about UI, like kind of a hierarchy and it seems like where we’re going with — You identified this well in these seven APIs. These are the seven API primitives that you're going to build your rich interfaces with. Much like whatever platform you’re your building for, you’re going to use hash maps.

[0:31:20.5] LR: Yeah, that's a really great analogy. Yeah, I think it hits it right on the nose. The way I kind of came about this is like I don't want like seem like I'm just this academic who said, “I think these are the primitives.”

In reality, it's React Native that kind of went through the hard work of choosing them, but React native is massive. There are like 73 APIs that React Native exports. When I started building applications with React Native, something kind of emerged from it which was the realization. I created a pretty expansive component library for Airbnb, like roughly like 100 components or something and I realized that at end of it, I almost only used those seven APIs. There was very little else that I used at the core, like React Native export level. Those APIs end up being view, text and image which are all React components that for people listening that are kind of used to the web, you can think of them as loosely corresponding to div, span and image.

Then the other like kind of really important UI primitive is style sheet which is essentially the layout algorithms. You can think of style sheet as a proxy for the Flexbox layout algorithm and as a way for defining styles that end up being passed into these core UI components of view, text and image.

Those are kind of these four pieces of UI that are really important. Then there is a — You need user interaction, right? At that point you can kind of describe static UIs, but you need some way for the user to interact. A powerful kind of abstraction around that I came up with was this idea of touchable, and touchable is like —I'm actually considering renaming it to Pressible, but it's just — In React, you can wrap things with this Pressible component and then they become interactible. You have things like an on-press and on long present and things like that that you can get out of it.

Touchable actually has an animated kind of interface to it. You get this kind of animated effect going from un-pressed to pressed and that that is part of a dependency to an API called Animated, which is one of React Native's core APIs that is a really really elegant abstraction around animations that has a declarative interface that works really really well with React. That's an API that I've also chosen to be included because I think animation is becoming more more of an important kind of primitive part of UI design and this does really a good job of it I think.

Then there's kind of like one last API which is platform which I don't even really think of it as a primitive. I think it's just a utility and it's just sort of a practical utility to be able to branch some logic based on platform, which is inevitably needed not certain times. That's one of the ways that React Native allows you to branch by platform, but there's another way which isn't an API per se but is just the way the packager in React Native works where you can have files with different extensions on them. I call them platform extensions. You can have a component, like button or something, and so you could have a button.js. If you want it to have a different implementation on iOS, you could have a button.ios.js. This is just a very simple way to take modules and have an interface that's the same across all platforms but with drastically different implementations that kind of allow the developer to very clearly demarcate those lines.

[SPONSOR MESSAGE]

[0:36:10.0] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today.

Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job. Maybe you want more flexible hours, or more money, or remote work. Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly passionate about technology.

Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge. Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:37:40.4] JM: Right. What's worth pointing out here is that at least today or when I last reported on React Native in detail, there's a number of instances where you're going to want to specify stuff about the platform that you're hitting. That's why it's not just like write your React Native app and it magically works on iOS and android. There's certain ambiguities to — When we talk about that event log, there are certain ambiguities where there's an event that is going to cause a change to the virtual DOM, that event might have some ambiguity on specific platforms. The idea of platform extension would resolve that ambiguity. You could probably explain this better than I could. If I'm unifying my app across web and mobile, what role do platform extensions play?

[0:38:37.3] LR: I think platform extensions are maybe the most important aspect of all of these. The primitives themselves that I've identified are themselves possible only because of platform extensions. I’m just using an interface that I want to make very clear and I think it's an important interface. Basically, I'm choosing that interface and then implementation for each platform is completely different in plantation, but the public APIs is identical.

That's a powerful way to build things like kind of the bottom of the tree, and so you create these common interfaces at the bottom of the tree and kind of — Then on top of that, you have this like pallets of cross-platform components that you can work with. Actually, it's really important at the top of the tree too.

A lot of people, when I start talking about this, their immediate gut reaction is like, “Well, I don't want my iOS app to look the same as my android or the same as my website,” and that's like totally fine. I think it's really important that this kind of flavor of cross-platform development is it's opt-in. You can only share what you want to share and not share what you don't want to share. You can have different entry points per platform and you could start off with two completely different code bases. You could implement both apps completely separately. They might be in the same repo, in the same folders or whatever, but they could have a completely separate, like zero intersection implementations.

Then you might realize like, “Hey, over here, we built this one component and over here we build pretty much the same exact component.” Because there aren't any — Because you're using the same stack, the same platforms, there's no issues with just deleting one of those files and using the one. You can just do that, and then you start to realize that, “Oh, actually, a large chunk of my application is the same,” but there may be really important differences that you want to preserve.

On iOS and android there are things like the navigational model is often quite different. People on android maybe want like a slide out drawer. People on iOS maybe wan some tabs. Maybe you want a bottom right to material UI, circular cerate button or something, but you don't want that on iOS. Lots of those decisions are perfectly valid decisions to make.

It's important to know that like that's all completely possible here. You can you can make things as different as you want or as the same as you want. The point is that there is no longer this artificial barrier from sharing code that's there only because you're using different technologies.

[0:41:53.7] JM: I saw a really good demo that illustrated this that you gave at that React Native Europe Talk that I'll put in the show notes, by the way. The demo was like a VR — Like you're in VR and there's just like what looks like a smartphone interface floating in VR. It's just like, "Okay, I could see the thrust there." Like, "Okay, VR is not exactly the same." You don't necessarily want the same paradigms," but might as well be able to slap your smartphone UI code into a VR app. There's no reason why that should be complicated, and React Native allows that.

This is interesting, because there's a lot of React Native stuff that's been written at this point and VR has not gotten super popular. It'd probably get popular eventually. People will want an easy way to port their code to it and that I can see React Native naturally. People who had written React Native apps at that point will have a nice time adjusting to that. I know React Native VR has been created already.

I kind of want to just — For a thought experiment and just to kind of hear your thoughts on this, like everybody is getting into AR. Everybody is rebuilding their AR platform. Apple released an AR kit already and eventually we'll have glasses that we can see AR components. I know this is pretty speculative, but I'd love to know what would be the process of — Just to like walk me through how somebody writes a bridge between React Native and a new platform, because when a new platform comes out — Or we could even be talking about React Native for cars or whatever that would mean.

What is the process for writing — For getting the bridge necessary to hit a new platform? I think AR is a good example, because AR looks somewhat like the platforms that we deal with today.

[0:44:06.7] LR: Yeah. It could be more or less involved depending on kind of what you're able to do on whatever platform that that lives on. If the platform that that runs on can run C++, for instance, then it's probably not — That in and of itself kind of saves you a lot of trouble, because, one; most of the JavaScript runtimes are C++ based. You could throw in JavaScript core and then run it on with C++. Then React Native now itself is built on a cross-platform C++ bridge. You can actually have the bridge implementation too sort of for free.

Then once you have that, you basically have — The environment that your JavaScript code is going to run on and then you have this like kind of message implementation. Then once you have that, basically what you would need to create is part kind of the world — What in React Native we call the native modules. You would want to create like an implementation of the view and of the text and things like that and. Once that kind of comes in, you need to start interpreting these commands that I was calling them earlier. React Native has some kind of interfaces that are already defined that you would probably want to implement as well. There's one called UI manager, and that sort of is the React Native like renderer on the native side that kind of starts instantiating views and things like that.

A big part of that in particular is the layout algorithm. Again, this is another thing that's kind of been done for you. You have yoga now, which is a C++ library that Facebook has put out which is the Flexbox algorithm. With that, you have to kind of wire that up to whatever view system you'd be using. If this was like a new platform, it might have a new way of laying out views. You would need to — Tell Yoga how to kind of traverse that view hierarchy and then things like measuring texts are really important there, is you need to expose a method of text measurement and like intrinsic sizes of views and things like that.

That's like in a nutshell the — That is like what the renderer. That is what React Native is, is kind of those things being set up. It's like a set of native modules. One of them being the UI manager, which is a really important one that ends up orchestrating layout with the usage of yoga. Pretty much all of that — Anything below that is like already done for you in C++ which most like environments that you're going to dream up like probably have some C++ capabilities, but some don't. The web is an example of one where we don't really have the ability to run C++.

You would want to implement those things on the web, but like Yoga for instance can be compiled to JavaScript via asm.js. That algorithm can be run on the web which this is how — This is how React VR works, actually, because React VR is actually browser-based. It uses WebGL and 3GS, but actually runs in JavaScript. The code that is in React Native, like running with native code, like Objective-C and C++ in React VR land is just JavaScript.

[0:48:14.9] JM: The web assembly route is where C++ runs. C++ gets compiled or transpired, I guess, to this small subset of JavaScript functions and then that code can run on the browser. You're saying that's how web VR works? I'm sorry. React VR?

[0:48:36.7] LR: Yeah. React VR doesn't use web assembly, but it could in the future. It uses asm.js, which is just a protocol that some browsers implement that if valid JavaScript is written in a certain way, then it will essentially execute it as — It will like kind of parse it as C++. I guess it's not C++. It will run it as bytecode, I guess, and it's like extremely fast. There's no dynamic types or anything like that in asm.js. If it de-ops into normal JS, then it would still be like a valid JavaScript program.

[0:49:25.1] JM: It seems like if the requirements for building a good underlying system that React Native can run on top of, the requirement to basically run C++, which is like any operating system. When we talk about like the future, I mean it seems like we basically got a new operating system and it's going to be a cross-platform thing and it's like, "Cool." All of a sudden Facebook came out with this thing that's going to sneak up on us and be like the new — Right now, React Native apps, it's just basically like having to form fit the current generation of mobile operating systems that developers are kind of fed up with dealing with these two platforms and having to deal with the web also and it's just like — It seems like React Native kind of cures all of that.

Assuming React — And I know you have high hopes for React in the future. I saw that quote about Guillermo Rauch talking about how we're going to exploring React Native and React for the next decade or the remainder of this decade at least. Have you thought about what is the optimal underlying substrate that React would be running on top of? Because surely it's not like this alien operating system that React has to bolt itself on to.

[0:50:56.5] LR: Yeah. It's important to not underplay the role of like the platform here still. I guess the actual implementations of things like view and text and image and things like that, that there's a lot there. We're actually — React Native is really bootstrapping itself on top of an already fully rich UI system that is UI kit on iOS and is like the android view system on android and is the DOM on the web. Those things are not simple and there's still a lot of work going on there.

I think it is a really compelling kind of line to draw in the sand where you could be a company that said like, "Hey, we're going to build as much as we can kind of above this line with this interface." I think that that could afford a company a lot of — It's kind of like a nice head. If you want to enter into a new platform, all of a sudden it's maybe not that hard. If you want to make a play in like the VR space and you actually didn't have to rewrite a bunch of your application to do that, but you could rewrite like a very select portion that you want it kind of utilize the new features of that platform. That's a really compelling story, I think.

If there's something new comes along that kind of changes things dramatically and looks like it could be the future but you're not sure, it's really nice to not have to feel like you're really clinging on to this platform of the past and that you could just build these primitives so to speak and kind of already have this pretty rich experience to start off with, like relatively small investment. I don't think that that's like the reason to use React Native or react primitives or whatever, but I think that that's like a really interesting aspect of it.

[0:53:14.2] JM: Okay. It's been really fun talking to you and we barely scratched the surface of this stuff I wanted to get to. There's a lot of other material people might have seen or heard about this thing, React Sketch app that we didn't get to today and I know you've also got — I'm sure you could have said some interesting stuff about GraphQL, but people can check out that stuff in the show notes and maybe we can do another show in the future. This is really fun.

[0:53:41.4] LR: Yeah, sure. I am happy to talk anytime about those things. There is probably another shows worth of stuff there.

[0:53:50.8] JM: Absolutely. Cool. Leland, thanks for coming on Software Engineering Daily.

[0:53:54.0] LR: Yeah, than you very much, Jeff, for having me. It's fun.

[0:53:56.7] JM: Okay. Great.

[END OF INTERVIEW]

[0:54:02.0] JM: Your application sits on layers of dynamic infrastructure and supporting services. Datadog brings you visibility into every part of your infrastructure, plus, APM for monitoring your application's performance. Dashboarding, collaboration tools, and alerts let you develop your own workflow for observability and incident response. Datadog integrates seamlessly with all of your apps and systems; from Slack, to Amazon web services, so you can get visibility in minutes.

Go to softwareengineeringdaily.com/datadog to get started with Datadog and get a free t-shirt. With observability, distributed tracing, and customizable visualizations, Datadog is loved and trusted by thousands of enterprises including Salesforce, PagerDuty, and Zendesk. If you haven't tried Datadog at your company or on your side project, go to softwareengineeringdaily.com/datadog to support Software Engineering Daily and get a free t-shirt.

Our deepest thanks to Datadog for being a new sponsor of Software Engineering Daily, it is only with the help of sponsors like you that this show is successful. Thanks again.

[END]