EPISODE 564

[INTRODUCTION]

[0:00:00.3] JM: Most new front-end web apps today use ReactJS. An increasing number of mobile apps are created using the cross-platform components of React Native. GraphQL, Facebook's open source data fetching middleware tool is being used by more and more companies who are finding that it simplifies at their development. Facebook's open source suite of technologies created a new developer ecosystem.

There is an increased demand for engineers who know how to build software with React, ReactJS and GraphQL. This was the reasoning behind Gabe Greenberg starting G2i, a developer marketplace of engineers who write ReactJS, React Native and GraphQL applications.

In this episode, Gabe, Lee Johnson and Chris Severns, all from G2i join the show to discuss React and other Facebook open source technologies, as well as the ecosystem around them. We explored the architecture of a developer marketplace business and how to scale a consulting company, which is not easy.

Before we get to the episode, I want to mention softwaredaily.com. Software Daily is a place to post your software projects you can get feedback, you can find collaborators, and Software Daily itself is all open source. The Software Engineering Daily open source community has been building it for the last year or so, and it's pretty amazing to see what's come together. Now we'd love to see what you are building, if you have an open-source application or a side project you've been tinkering with, or an academic computer science paper that you want to get feedback on, then come to softwaredaily.com and post your project.

Software Daily is about cool projects, new ideas and creativity. If your project is especially interesting, we'll send you a Software Engineering Daily hoodie, or t-shirt, or even have you on the podcast to discuss what you're building.

With that, let's get on with this episode.

[SPONSOR MESSAGE]

[0:02:10.4] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous deliver to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[INTERVIEW]

[0:03:32.0] JM: All right, I'm here with the G2i team. Gabe is the CEO of G2, Lee is the CTO and Chris is the hiring manager. Guys, welcome to Software Engineering Daily.

[0:03:42.1] GG: Thanks so much for having us, Jeff.

[0:03:43.3] LJ: Yup, thank you.

[0:03:44.2] JM: I'm excited to talk to you. I have used G2i as a customer. I have gotten contracting work done with you. I've also worked personally with some of the engineers who have gone through G2i. I'm friends with Caleb Meredith, who now works at Facebook and he's

actually done a show on Software Engineering Daily. He interviewed Brendan Eich about JavaScript, and I know you guys are deeply familiar with JavaScript.

I know that's why you led to starting a business around React and React Native contracting. This actually is a topic of interest to me. I've done a number of shows covering the contract engineering space both because I think contract engineering is an interesting route that some engineers choose to take because it gives them a lot of independence.

Also, I think that contract engineering is under-utilized as a way to build minimum viable products, or to augment your workforce. I just think that the idea of contract engineering is under explored as a technical route. You even have, like I saw a tweet from Michael Seibel from Y Combinator recently, where he was saying that he still sees Y Combinator applicants that have contracted out the first version of their software and he sees it as a negative. I wholly disagree with that notion, and maybe we can get into that more, but I think that speaks to the reductive attitude that a lot of the industry has around contract engineering.

I'm hoping to discuss contract engineering and React and React Native today. I guess, to start us off what are the trends that are changing the adoption, the increasing adoption of contract engineering? You've got more clients that are purchasing contract engineering, you've got more engineers that are going in that direction. Why is that changing?

[0:05:42.8] GG: That's a great question. I really think to start out, it's just becoming more accepted and it's becoming more just that the way things are headed. The reason being, being that we're a distributed company dealing with mostly distributed engineers comes down to bandwidth, internet bandwidth as simple as how often are we doing video calls? How easy is it to do video calls with somebody?

We have a core team member in Nigeria, we have engineers all over Europe, engineers all over South America, and five, six eight years ago, me doing video calls with somebody in Nigeria, or Kenya, which I've spent a lot of time living there, it wasn't really possible to do regularly, consistently with the bandwidth issue.

Just video calling, whether it's through Zoom; we use zoomdaily.co, which is the Y Combinator company. Whatever your method is having better bandwidth accessible around the world really matters. It just allows certain touch points that you just wouldn't have via e-mail. Then real-time messaging, Slack really has become a way that we can work remotely, work distributed, and have some organization involved in the way that we message back and forth.
Also just a tight remote culture, you know having companies that are actually thinking about that first. When we've interviewed candidates, they've asked what does look like daily to be on your team? When you're part of our core team, or the internal team at G2i, Chris jokes that he sees our CTO Lee more than he sees his girlfriend.

[0:07:18.3] CS: It's scary. It's how true that is.

[0:07:20.5] GG: Yeah, it's scary. It's scary true. Lee's in Mississippi and –

[0:07:24.5] CS: I've never met Lee in real life.

[0:07:26.0] GG: We've never met him life, but we know each other really well. That comes through FaceTime that it comes through regular FaceTime stand-ups that we do. I think probably Infinite Red, their React Native shop, which I love Jamin and the team. I respect them greatly. They have an always-on video channel.

They've rigged some things together and built it custom, so that when they have – when they're taking lunch, sometimes they can jump in front of that camera and have it a non-business, or non-development type get to know you, the water cooler conversation. I think a lot of these things together, I have made remote contracting, remote engineering, whether as part of a team, or if you're a contractor, just more at the forefront.

[0:08:14.1] JM: I've had some conversations with different companies that do contracting of different types. I think the most recent one was with Gigster. They have a high-end contracting model, but one of the things I discussed with Roger, if I remember our conversation correctly, was this idea that the software that we build these days, while the components that we use, React has literal components but there's also things like AWS and there's some well-defined best practices in NodeJS, for example.

This is one of the reasons I think that that contracting has gotten better, because there's just more information around how to code properly, and whereas in the past, if you would have contracted a project, you might have ended up with this rat's nest of code that was not very well written. If you wanted to take it over and scale it and build it out, then you might have all kinds of technical debt from day one.
It feels like today, contracting is a little bit safer, because first of all you can build products, you can build a better product with less code. Second of all, the code that gets written is generally of higher quality. Do you find that to be the case, or do you think I'm – do you think I'm a little overly optimistic about the general state of contracting?

[0:09:40.5] LJ: I could touch on that briefly for, and I know that for us specifically we purposely put every developer through a set of tests, and I guess for lack of a better term, so that we know when they're on a project, they are following those best practices, those React best practices of component-based stateless architecture. There really isn't any technical debt, and that's why we focus in an area like that.

I think why it is growing is because you can't get it – most of our shorter length projects start with that. They start with your contractors coming in, but then they take that product and what they've learned and they continue to use it and build on it forever, because it's built from the beginning using best practices. We make sure that that is part of, but now it's not just some technical or language specific, but it's best practices also.

[0:10:35.2] JM: You guys run a React development team. You have jobs in ReactJS and React Native and you find clients with those jobs and you have a team of engineers that build those products. How did you get to this point where you ended up building a development team exclusively for ReactJS and React Native projects?

[0:11:01.8] JM: Yeah. I mean, well it wasn't on purpose, I'll tell you that much. I started out in web development and on the frontend basically in Africa and Kenya. I was doing missionary work with street kids and I came back and I was – I had a technical background and charity that they asked me for a website. I didn't really know what I was doing, but I figured it out. That's how it started for me.

Eventually, G2i was just local-focused here in South Florida and our projects were getting bigger and bigger and we were starting to outsource to engineers in Europe to help us across the stack, and it was Sam Breed of Quick Left at the time, he was the CTO of Quick Left, he pushed our team into using React.

The team loved it after about a week of complaining about JSX and a couple other things at first, but they get over it really quickly when they saw the benefits. We were building a really big EMR project electronic medical records in the healthcare space, and the client had some issues legally with their competitors and they shut down the project overnight.

I was newly married, had a baby on the way, and we had a team in Europe and we didn't know what to do. Really we started connecting to Y Combinator startups, really not even on purpose and other startups and they needed augmentations. They didn't need someone to go away and come back in 30, 69 days of here is the finished product, or check-in every once in a while. They needed engineers to get to demo day at Y Combinator, to scale after they got funding, before they could hire the team.

It was connecting with Caleb. Honestly, Caleb Meredith he's now Facebook. I hired him when he was barely 18 and he came in and he was the first person to say, “Hey, let's move away from this dev shop thing and let's just vet engineers technically and match them with companies.” He started that process and we've done it like that for about two and a half years now. We also provide vetting and sourcing for on-site roles in New York City, San Francisco, Seattle and a couple other cities across the states.

[0:13:06.1] JM: People come to you. Was it that vetting thing? If somebody is looking for a full-time React engineer and they don't know how to do the interview process appropriately, they outsource the interviewing to you?

[0:13:20.5] GG: They outsource everything, in the sense that most of our companies are just – they’re larger companies, they're moving fast. The biggest disparity between supply and demand is in the React and React Native ecosystem. We know intimately our community and

we love our community we want to give back to our community, but we also can find engineers and then technically vet them.

In the end, these companies get a very technically focused profile with a code challenge score, actual code written, technical interview score and notes, some video as well. We're starting to add of our interview process, and it just speeds up things on their end in terms of making decisions.

[0:14:03.3] JM: This is the business that in the past would have looked like a business that is not scalable, because it looks – it's like a matchmaking business, or it's a consultancy business and those are typically thought of as these less sexy, less scalable businesses. My sense is that because of simply the quality of the tooling, whether it's Slack, or Asana, or Gmail, or just the fact that everybody's online all the time so your responsiveness is higher, things have changed where businesses that previously would not have been scalable, actually are scalable.

In addition, you just have an increase in liquidity and quality of engineers and the leverage of engineers that your transactions can be of higher value per engineer. Do those things hold true? Have things changed where in the past, this would have looked like an Accenture type of business, where I think people associate Accenture with – not to speak badly about Accenture; it's a great business, but I think people associate with it as a company that's very big and has a lot of process associated with its large consultancy. You can now have a consultancy that is scalable without a large core workforce that is mediating that consultancy. Has it become easier to build a scalable business with this model?

[0:15:34.6] GG: It's definitely become easier, but it goes back to know our market isn't as sexy as some other markets. We're okay with that, but given that the remote culture is stronger, yes it's easier, it's been done before. There's other companies out there that have scaled very large and we're also not just trying to scale large, but we're niche at the same time, and we're really focused on doing the best in the React and React Native ecosystem that we can possibly do, then going out into the other areas that we can scale into.

We want quality first, before we get to the scale. Ultimately, it is about finding the best possible engineers you can, and also helping some of the mid-level engineers grow, which is a part of

our process. It's ultimately about matching them to really great companies that place a high value on engineering.

Then there's some of the infrastructure questions, how do we build our infrastructure? Do we do it from scratch? How much do we build? How much do we buy? Those are really interesting questions, which we're starting to answer now. I never want to make the main thing not the main thing, so we can get lost in building our own software. Then five years down the line, we're actually behind.

[0:16:52.1] LJ: To add to the scaling point, we are very focused and we're very technical too which is a little bit different. We drink the React Kool-aid, or the technology Kool-aid and I've spent many, many years as a senior consultant for Deloitte. I've definitely lived that world and you could scale that by bodies, right? You had people, but the way that we're pursuing this is scaling it by the technique and technology and focus in a specific industry.

[SPONSOR MESSAGE]

[0:17:30.5] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:19:05.6] JM: React, why do people care about React? Why is it easy to find customers who specifically want React if they're focused just on their own end product, you would think that they would be coming to you and saying, “I don't care what front-end framework or stack you're using, just build this product that we have a spec for.” Why do people care about the programming stack, when they're coming to you building a product and why specifically do they want React?

[0:19:38.1] LJ: I think a lot of it goes back to your initial observation and that you don't have to throw the code away anymore, like you're building something. For me it's the component architecture that I love, then that makes it so reusable and it leads people and their reusable dry sense more than other frameworks. Under the very first project I did with React, it's actually React Native, but similar idea.

The product owner had the same question and thought it was silly, but after he saw what we accomplished in the short amount of time and how rapidly we were adding to that, he was sold and giving talks at meetups about how great React was. I think it’s one of those things that once you see it makes sense, and then when you add to something like React Native, but you had this – if you build on this ecosystem for your primary web product and you can take that knowledge directly to mobile, it's just for someone who's hiring people, it just makes sense. You're looking for the same engineer type for both areas and then you can reuse all these things.

[0:20:47.9] JM: I started covering the React stack of technologies near the beginning of Software Engineering Daily. Why did the React project gained so much traction?

[0:20:57.8] LJ: Yeah, I guess again for me it was the – when it came out, it was heresy, right? You were combining your logic, your layout and your style in one thing. It was totally against what everybody else told us we should do, and then NBC was the way the pattern of future and what everything is supposed to follow.

It came out with this just totally off the rails approach at software development and ideas, and people bucked against it, but then they started using it. Just like the example I gave a minute ago, you start using it and just the light bulb goes off and it makes sense, and then that spread like wildfire. Popular people started adopting and contributing to it and talking about it, which encouraged other people to try it. When they try it, it's this new way of thinking became refreshing for me, from my personal experience.
I think that translates to all developers who have struggled and have been frustrated with this whole crazy JavaScript ecosystem that now there's this thing that's a lot like object-oriented programming all of a sudden. It's just nice to use and it's nice to write code in it, and just the fatigue is lower.

[0:22:09.2] JM: When I started covering React, it seemed very exciting, and it seemed there was the potential for React and React Native to take over all of mobile development eventually, like where it would be web and mobile development, it would all be this integrated development experience where everybody is working in React. I don't have the sense that we're quite there yet if that is an eventual future, but where are we in the timeline of React developments? What are the biggest new React developments happening today?

[0:22:42.2] LJ: Well yeah, I mean, you're right on the adoption. For whatever reason, React Native adoption has been much slower than React. Well of course, it's not as old, but still it's just in the velocity of it. It has been slower. I'm honestly not sure why, because I'm obviously a huge fan of it and have used it in production for many things and it works great. I don't know if it's just – I don't know Gabe if you have some ideas of why, but the adoption for that is not as rapid.

[0:23:10.0] GG: I think part of it, just a speculation, but again going back to there's a huge gap in terms of supply and demand for React Native developers, especially at the mid and senior

levels. Lee, maybe you can talk to this a little bit, but if you're having a hard time finding mid to seniors in this technology, it can be scary because what happens when you hit problems with optimization, what happens when you hit problems where you need to introduce native code, these exceptions to the rules that you need a little bit more of a senior engineer to tackle.

If you have a have trouble finding that senior engineer, it may back you off from React Native, because it's a lot easier to find a senior iOS, or senior Android engineer and get them onto the team and work in a native code base that's familiar to them. I mean, have you had some of that experience Lee, even before working with us?

[0:24:07.6] LJ: Yeah, that's a great point and even just to add to the why of that is no web application developers already know JavaScript. Going from angular to React it's not an entire mind shift, but mobile developers do not come from that world. They come from Java or Objective C.

We are talking about a pretty dramatic mind shift and it does help to have some of that Native background, if you're the only engineer on a team doing a React Native project. That's probably the biggest hurdle. It’s just two different worlds on the mobile side.

[0:24:39.5] JM: Here's my thesis, and I could be totally wrong, but in the earliest days of mobile development, I remember this series of technologies like Appcelerator, and I don't know what else there was. There were these other tools that were – they promised the cross-platform dream and they never worked as well as they advertised. I feel like people got burnt out and they just decided that they had to do native development. There is this hangover from the experience of trying out these tools that didn't work. Am I mistaken there?

[0:25:20.4] LJ: No, that’s definitely happened. You had Appcelerator, you had Zoomer and you had PhoneGap, you had a lot of those things. Ionic, a lot of those things that we're doing the same thing in the early ones it did – sometimes it didn't even say anything. You had to steal right separate components for Android and iOS in the early days.

It didn't save you a lot of time and it has a lot of overhead. I definitely think people had a bad taste in – even for people who didn't write code for a living would tell you that it will never be as fast as Native.

[0:25:49.2] JM: What did React Native do that was different that made it more amenable to actually shipping stuff and fulfilling its promises?

[0:25:58.2] LJ: Well, it's so the way it's built obviously is a little bit different. They opened up the RIC JavaScript runner that you could have access to that was a lower level, so it ran faster, but also the way React Native works is that there's a native component married to all of your JavaScript stuff, so it's actually running Native code. Things like PhoneGap obviously it wasn't, it's just a webview.

Appcelerator definitely did some of that early on, but it was early on. Then I think to another point that Gabe points out all the time is look at the device that you're holding right now. It's massively more powerful than the device from five, six, seven years ago. I mean, that alone is a big difference, that everything runs faster.

[0:26:42.3] GG: That's a big reason why electrons become more popular as well. I see larger companies like Airbnb, they're using React Native, but they're not using 100% React Native. Most their code is Native and where they were using web views before, maybe they're using React Native there. There's different levels of adoption as well on these larger companies.

Discord for example, they do TeamSpeak for gamers, really well-known and they host our Reactive Flux account as well, so they're great for open source. Discord they've chosen to not use React Native for Android. They just didn't feel that the performance was good enough on Android, but they use it for iOS mixed in with their Native code and it creates a great performance solution that I use daily.

I know of one other company that's moved away from React Native after having a pretty sizeable codebase for some performance issues, some issues maybe with resources and the CTO did change over. In terms of performance, we really don't have a one-to-one case study. I don't think that really exists of a production level application that was done in say, Swift, and

now is in React Native. We'd love to create one. We're going to talk to one of our clients about doing that, but we don't have any really truly objective performance case studies.

[0:28:05.5] JM: That incremental adoption is such a big selling point for React and React Native. You don't have to jump with both feet in if you're not ready to. You can just start to select some components, or some screens that you want to migrate to the technology just to try it out, just to get a taste of it. That's such a big deal, I think, where you don't have to, like you're saying, build an entire clone of your application in React Native to really just understand how does this work. Is this going to be useful for us? Is this going to be productive for us?

A lot of companies have started that way and are still running that way, picking and choosing where they need the most performance. They stay in the Native and where they're willing to make some trade-offs, or where the performance differences are negligible, they're moving to React Native in order to gain that shared codebase.

[0:28:54.0] JM: Gabe, I think you mentioned electron is growing in popularity partly as a byproduct of the advance in device quality. I thought electron was mostly a laptop side thing, or a desktop – for desktop applications. Does electron have an impact on mobile applications?

[0:29:16.4] GG: Not that I know of, but I was really referring to the same idea that our laptops have gotten faster over the years. Whereas, we didn't have the same resources, CPU power, memory maybe six, eight, ten years ago and developers would have pushed back to, “No, we have to do Native for desktop.”

Now it's like, “Okay, we're willing to make some of those trade-offs so that we – because we're responsible for now not only the web, but a desktop version and we're responsible for a mobile as well.” Then something React and the ecosystem that surrounds it makes more sense, because of the increase in technology.

[0:29:52.4] CS: To that point, even with the development side, I have a MacBook Air that's a couple years old now and I'm able to run the React Native simulator on it without any real performance issues. The only time it really drags is when I have Slack open as well, because I'm in way too many slack channels. It's actually quite shocking how good the simulator runs on

that MacBook Air, which you could look at and say, Well, that's really underpowered to do this," but it's even great for the development process as well.

[0:30:22.2] JM: Well, that's reassuring. I done some recent iOS development and the simulator was also pretty good for us, at least a simple application, but it's good to hear that that even for – maybe for a more complex development the quality is good. I guess, with the electron stuff electron, is it a resource hog?

[0:30:42.0] GG: Yeah, for sure. If you're writing the same desktop app natively, it's going to take up less resources. That that's generally speaking high-level. Of course, there's things you can do to optimize everything, but Slack runs on electron. I'm sitting here in get station, which is great and as all my things I normally open up in Chrome that's electron. Discord uses electron, I believe.

[0:31:05.5] JM: What did you call – what is that, get station?

[0:31:07.4] GG: Yeah, get station. G-E-T.

[0:31:09.3] JM: Get station.

[0:31:10.1] GG: Get station, yeah.

[0:31:10.7] JM: What is that?

[0:31:11.5] GG: It allows me to have about 300 different apps that I use, but I have Slack, Todoist, Gmail, HubSpot, Discord, Trello, Paper, Level, AngelList, and so on and so forth in one browser where all my notifications are easy to change side panel. That's just allows me to also bookmark the different apps that I want to use and the different pages within those apps that I want to use. It's made for modern web apps, and you get lost in 50 chrome tabs. It solves that. I think was Michael Seibel from YC that posted it up, and I've been using it ever since.

[0:31:51.9] JM: Interesting, because I'm in that mode where my MacBook is basically an operating system that is basically has a Chromebook on top of it, because everything I do is just Chrome tabs.

[0:32:06.0] GG: Yeah, exactly. Yup. That's our life.

[0:32:09.3] JM: Yeah. Talking a bit about the changing nature of the React development environment, there's a few things I wanted to explore, so React has a newer feature called asynchronous rendering. Explain what async rendering is.

[0:32:24.0] LJ: Yes, it's so obviously very new. It's pretty big change in the fact that the lifecycle, which is really nice in React is changing. The component will mount, component will receive props, component will update, they're going to mark them as unsafe, then mark them as deprecated and then remove them. These are pretty pivotal things that people use and they're going to replace them with a get derive state from props.

Asynchronously, a component can receive its props and state changes and then render when it needs to, instead of kicking off all these events basically, or listen all these events. A common use case example I think is if you were loading data right and you called that on component will mount or something, the state is going to do things before that data comes in. This is just by nature, or the way that the life- works, this thing's going to kick off and happen and update.

Then today is going to come in and those things are going to reap, update and kick off. With async, you can actually tell the whole components, don't do anything, until I get this specific prop, or this specific state change, and then render. It's a really smart approach to it, streamlined whole render process but doing it an asynchronous way, so that it's not based on all these steps and it's just based on a a single asynchronous stream, if you will.

[0:33:49.5] JM: Does async rendering have a significant change to how a React developer writes their code?

[0:33:56.3] LJ: Basically around those lifecycle hooks, it will change that. Really with modern best practices, those aren't used as much. For our devs, we really push stateless components

and some manager container, but this will still improve like you can really view the incoming props and state change and render.

It will cut down on your render cycle. That may be a little bit more, if you really want to be performant and maybe use some more checks that you have to do, that you normally wouldn't do, and you're going to be missing some life cycle events that you may be used to having. In that way, it will be different.

[0:34:35.2] JM: There are some language tools, like reason and flow and typescript that make it easier to avoid compiler errors in React. Do the developers on your team have a preferred way of getting type safety?

[0:34:52.1] LJ: I think each person has their preference. Typescript is a pretty dramatic change. It's very different and it compiles everything and it's like a whole another language within a language. I personally like it, but it's definitely a shift that everyone has to be on board with. Flow allows you to adopt it more gradually.

The negative I've seen to that is if you give a developer an option to make their job a little more difficult, they're not going to take that option. I mean, flow is optional, it tends to just get ignored. It just really comes down to how intently you want to focus on type-checking. A lot of the guys we interview bring up these same points, reasons very new, so they haven't gotten into a lot of that yet for their work, but a lot of them I actually referenced prop tights, which is another good way to try to force that type safety is to use prop types.
When prop types were actually removed from the core React to a different library, I think people will soon, "Oh, well we don't have to use it anymore, because it's not in there anymore, and it's just not as you use as often as used to be, but that's another good way of doing it."

[0:36:04.5] JM: What about server-side rendering? I know you can make your React application get rendered on the client, or on the server, and there are advantages to rendering on the server side, such as if you render on the server side then it puts less burden on the client. It might have some advantages for caching. It might be easier to get your page indexed by Google, for example, or might get your meta tags, like if you're doing social sharing, then you need the server-side rendering to happen in order for the meta tags to render properly. What's the state of

server-side rendering? Do React apps typically all get server-side rendered these days and is it hard to implement that?

[0:36:49.9] LJ: Well, I think to the points you just mentioned they're all very good ones. The initial load time is also good benefit for just pure server-side rendering. The thing it comes down to, if you need those things, right? If you need those things you just mentioned, then it is something you have to pursue. I would say it's not used as often. It's not in there as often as it's way more often than it is, because create React app is very puffer starter and it doesn't include that.

That's how people get started and then they typically add server-side rendering later if they need those things that you mentioned. It's not difficult. It's a simple express with middleware application, but to me – I mean, you get those benefits, but then you can go a little bit further with code splitting, which allows you to then load certain components or groups of components later, instead of downloading an entire bundle at once, which again speeds up the whole process and experience. It gives you a lot of – for large applications, it gives you options and choices of optimization, speed, bundle, sizes things like that.

[SPONSOR MESSAGE]

[0:38:03.6] JM: Users have come to expect real-time. They crave alerts that their payment is received. They crave little cars zooming around on the map. They crave locking their doors at home when they're not at home. There is no need to reinvent the wheel when it comes to making your app real-time.
PubNub makes it simple, enabling you to build immersive and interactive experiences on the web, on mobile phones, embedded into hardware and any other device connected to the internet. With powerful APIs and a robust global infrastructure, you can stream geo-location data. You can send chat messages, you can turn on sprinklers, or you can rock your baby's crib when they start crying. PubNub literally powers IoT cribs. 70 SDKs for web, mobile, IoT and more means that you can start streaming data in real-time without a ton of compatibility headaches. No need to build your own SDKs from scratch.

Lastly, PubNub includes a ton of other real-time features beyond real-time messaging, like presence for online or offline detection and access manager to thwart trolls and hackers. Go to pubnub.com/sedaily to get started. They offer a generous Sandbox to you that's free forever until your app takes off that is. Pubnub.com/sedaily, that's P-U-B-N-U-B.com/sedaily.

Thank you PubNub for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:39:46.8] JM: On the Software Daily app that we've been building, we use view and server-side rendering for ViewJS has been not trivial for us to implement, so we haven't really gotten there yet. Instead, we are using this thing called prerender.io, which is I guess, a way of it lets you do client-side rendering, but it renders on the client then it caches it in prerender.io and then you can just load the page from prerender.io.

It's this nice caching CDN middleware thing that I thought was pretty cool when I found out about it, because it lets – if you're having trouble implementing server-side rendering, you can just have a middleware client do that rendering for you and then cache it and then read from the cache. I thought that was pretty creative way to shortcut that.

[0:40:38.9] LJ: Yeah, sounds really cool.

[0:40:39.8] JM: Talking a little bit more about React Native, how easy is it for people to take React components off the shelf these days and have those React components work on web and on React Native? Has it become this easy Lego block experience of putting together UIs, where you can just look through an index of React components and build your cross-platform application?

[0:41:06.8] LJ: No not in the cross-platform way. Now when we approach a build, we do approach it in that Lego way for that platform. If you're building a mobile application, we most definitely build Legos components separate and stateless and then you can definitely use those Legos to build your screens in a much more rapid rate, but it's on that platform.

For the multi-platform, the biggest use case and I know, Gabe can give some details on the specifics, but there's React Native for web. It tries to do that gap that you mentioned. For me, the important thing to remember about Facebook's mantra is it's learn once, write anywhere, right? The whole Java saying that write once, run anywhere worked if you have all the right jars and you installed it right and you have the right j2ee installation and the JRE.

It was a problem, but Facebook's approach is really once you learn how to do this, you can replicate it either way easily. That's really the intent. It wasn't like the Java write once and run it anywhere, that wasn't their mantra. It was learn it, and then you can write code like this in multiple environments. Gabe can give you a really good example of React Native web and how customers are using that.

[0:42:28.2] GG: Yeah, I mean React Native for web was written by a Twitter engineer, for Twitter light for the web application on mobile devices. It's been used by – Major League Soccer uses it, I think. Who else uses it production? Sony uses it. I mean, Major League Soccer uses React Native for web – the web for desktop on their React Native apps, as well for mobile, so they're able to have a much more tightly integrated and shared codebase, because they're using React Native for web. There's caveats to using React data for web; routing would be one of them. Yeah, it's not widely used, but I think it will become more and more popular.

[0:43:10.4] JM: To be clear, React Native web allows you to reuse a component for React Native iOS or React Native Android on your web.

[0:43:20.3] GG: Yeah, or vice versa. Yeah, yeah. You can share them, because you're writing React Native. Yes, exactly.

[0:43:26.7] JM: The difference between a React component and a React Native component is what exactly? If I'm writing something for the web with just classic React, ReactJS, I write a component for ReactJS, and if I was using a React Native component, what's the differences there and the similarities?

[0:43:46.3] LJ: Indeed, differences are what those things actually reference, right? In React Native, you would use – you were rewriting JSX just like you would in a web application, but

instead of dev for example, you would use view, and that view is actually directly tied to a Native thing; a view and Objective C, or in [inaudible 0:44:06.8] in a 10, or whatever. It actually references a Native piece of code. In web, obviously it's referencing [inaudible 0:44:14.7]. They're representing two different things.

[0:44:19.9] JM: If you wrote a component for React Native iOS and you ported that to the web via React Native web, what's going on in that porting process?

[0:44:30.7] LJ: Yeah, I haven't dug into the specific code, but think of it in the fact that that view references a React Native, a Apple view object. For the web that view almost likely represent a div, right? There's a mapping of what would be a view, what would be a picker in iOS as a select box in web.

Honestly, the components for the NATO, for HTML and the Native, they're similar, they're just call different things, and they're rendered differently because they run in a different environment, but they're very similar what they hope to achieve.

By mapping it one or the other based on a targeted platform, you should be able to use one code for different things. Now of course, there's different properties and functions for others that – that's what React Native for web does. It handles those for you. It does that navigation, or that split for you.

[0:45:23.0] CS: I know this isn't the only project that's trying to pull this off. Leland Richardson was working on React primitives. There's a couple other ones I've seen, I heard the most about React primitives, but I definitely think they've hit a wall, once you get outside of things like view button very low-level, simple components, you start to get into complexity in terms of how are we going to render this across all three of these platforms; iOS Android and web. It really does get tricky at a certain point.

I think Lee, your point about Facebook was all about learn once, write anywhere. Everybody kind of – or not everybody, but there was a lot of momentum towards, “Oh, this means we can write once, run anywhere and that's really not where we're at yet.” Although, I think it would be great to be able to truly write React Native once deploy to web, Android, iOS, VR wearables, all

the different experiences that are pretty still nascent and all the ones that are coming and aren't even here yet, that would be a really great selling point if we can start to figure some of those things out.

Just another team that's using something across all platforms that's React-centric is Microsoft. I forgot that they developed their own version of React Native for web. It's called React XP and while I don't know a ton about it, I did talk to their team lead and they're using it across desktop, Android and iOS for all their applications are in React XP.

[0:46:53.2] JM: The other technology I wanted to explore a little bit of was GraphQL. GraphQL was one of the suite of technologies that Facebook unleashed in this swath of open source developments. There were integration – I mean, there are integrations between GraphQL and React, but it's not like a tight coupling. People use GraphQL in other contexts and I've enjoyed covering GraphQL.

It's one of these technologies where I can't quite grasp the speed of adoption, because I will hear some people say GraphQL has really improved our stack, or we set up GraphQL from day one and it made – like I talked to my little brother the other day and he set up GraphQL from day one on a project he made and he said it just really helps shuttle him into the right way of thinking about data models.

Although I think, GraphQL is hard to understand for some people. I think there's the burden of setting up a GraphQL server that is the interloper for your queries, the interpreter for your queries. What is the adoption like for graph QL? Is it overhyped? Are people getting a lot of value out of it? What are you seeing from the client base in terms of GraphQL?

[0:48:11.0] LJ: Yes. I've been surprised actually in how often we see it, especially with new projects or new applications. Really have been surprised it comes up a lot. On the opposite side of that, I've been surprised at how little we see it for existing APS, like people not trying to integrate a piece of something in their existing API with it. I haven't seen a lot of that.

It's weird. For the startup culture, it seems to be really, really popular. For the existing rest guys, it's like they're – I guess, maybe they don't see the point. They probably have been through that

pain already and just stick with what they have. For rapid development, it's obviously quicker. If you've been on a team for the longest in a period of time and you're developing an application, maybe you're on the frontend team, that whole dance of deciding we need this piece of data, told me to the back-end team about getting that piece of data, then I'm telling you that you don't need that piece of data and how handshake and figuring that out, finally adding it to a rest endpoint or a new rest endpoint and then getting that to your application.

I mean, that's the process. With GraphQL, you're basically handing access to all of your permitted data points with a configurable query language, so you're allowing the front-end engineers to get the data that they want and that they need without having dependency on the back-end guy to add that.

In addition to that, they can ask for exactly what they want, so you're not downloading this humungous data object to render somebody's name and a phone number. You can get literally just the name and phone number, so your request is smaller so therefore your app load time is smaller and just all around it's a more efficient way of handling data.

[0:50:06.6] CS: To that point, we've been working on some internal tooling and an application that Lee's been working on, we're interacting with an API where basically there's a web hook set up. Off of that incoming data, we pull an ID, we hit an endpoint that we get a big blob back that all we really want is another ID to hit another endpoint, that all we really want is another ID to finally hit the endpoint to get the data we wanted.

You're making all of these network requests. I mean, it's really wasteful in terms of overhead. It's not really a big deal right when you have great bandwidth, but there are plenty of times where my bandwidth isn't so great. Even in South Florida where I live, there are plenty of spots around my house where I can't get any phone signal and making all those network requests would just grind anything I want to do to a halt.

The ability to just have one endpoint that knows how to resolve the data in an efficient matter and you're not making all of these network requests just to get the final information that you wanted, you're not throwing away tons of data and being wasteful about it, I just think that's such a great selling point.

[0:51:16.2] LJ: To any listeners that do have that existing rest and PostgresQL database or whatever they're using, I just hope they understand that GraphQL doesn't do anything. It doesn't touch your – it doesn't change your PostgresQL. You continue to use all of that stuff and simply add this new object, whatever it may be; customers, or users, or people, or products as a little layer on top of that. It just talks to your database. It doesn't change anything on that end.

Now people are definitely resistant to change on the database side and the data layer as they should be. That's very important. In most cases, that's your business, and so it doesn't require you to use any certain database, it doesn't change that. It's just simply an access layer on top of that.

[0:52:03.6] JM: Right it's a veneer. There's a company I worked at one time that had a ton of legacy code. This was one of my first introductions to how enterprises refactor legacy code. By refactor, what they often do is paper over it with a new API. That's fine, because you never want to cut out the giant rat's nest of code. You never want to cut out the big ball of mud. You want to paper an API over it that is simpler, that is easier to interact with, that is perhaps more efficient and. It's funny, because that's essentially what GraphQL does. Maybe they could have called it like veneer or something, or paperover.io or something.

[0:52:54.3] LJ: Lipstick. It's just lipstick.

[0:52:56.0] JM: Yeah, lipstick.

[0:52:57.7] CS: LipstickQL.

[0:52:59.9] JM: There you go. Yeah. I think honestly, I wonder if GraphQL was the wisest marketing choice for what the actual – what the product is, or if it would have had better adoption if they would have called it something different. But who knows? Who even cares, because it seems like the adoption rate is going just fine, even if it's a sleeping giant, it will – It seems it's so popular among the people who actually use it and particularly the power users, that it's an inevitability that it is going to be a widely adopted product eventually.

[0:53:35.0] JM: Yeah, I agree. You have Starbucks, New York Times, Pinterest, Intuit, Github, Major League Soccer. I mean, there's some larger companies pushing it forward as well; Apollo is amazing. I think, I agree. It's a matter of time. I think lipstick.io, lipstickQL.io is – we'll be working on that for you.

[0:53:57.1] JM: You guys need to just fork it and then rebrand it and then call it like G2i product. I mean, that's like the Bitcoin cash model, or some of these blockchains where they just fork Bitcoin and then rebrand it as something else and –

[0:54:14.2] CS: How can we work blockchain into it?

[0:54:16.5] GG: That's the key to success.

[0:54:18.5] JM: There you go. There you go. No, you actually just replicate everybody's schema on a distributed replicated ledger so that nobody actually has to run the GraphQL server. You have it distributed across a blockchain. There we go, all right. We have the ICO, it'll be in three months and the pre-sale begins today. Anybody that wants to buy lipstick tokens can just send me an e-mail. Yeah.

Anyway, so okay we got to wrap-up. I'm just a little bit interested still in the business model just to close off of G2i, because I think there's a lot of people out there that are thinking about what business to build. Definitely engineers listening that are trying to build a product, or they're trying to figure out how to start their own business. I think one thing you can do to get started is obviously start a podcast, if you have no idea what other kind of business to build, that's one avenue.

Another avenue is consulting. You can build a consulting business and maybe that consulting business can evolve into a product, or a marketplace. I think that's the evolution that G2i has gone through and it seems it's been very satisfying for you guys. I'm guessing that it is beginning to congeal into more of a vision, so I'm wondering there are these varying big outcomes for consulting style businesses.

You have companies like Fog Creek, or ThoughtWorks that have started to build products, you have Accenture as we mentioned earlier, that way you can scale up the consulting business by building internal tools and managing that business very effectively. How do you see the business of G2i evolving towards the future?

[0:55:56.4] GG: That's a fantastic question. One I've done a lot of thinking of on and been sharing the vision a lot more often with the team here and others that were hiring. I think it's a mixture of building products. I think we have some product ideas for sure, but our main focus right now as a company is building internal tooling to help automate some of these processes that that we want to automate, so that we can grow and scale and become a larger company.

A big part of my thought process is founders are always optimistic, right? We have that bias. We need to be get through the ups and downs. I've never done much thinking about what happens if we grow to a 50 or 150 million dollar company? What happens at the end of that? Or what happens if we don't make it and we fail? But we're in business for five years.

I've been thinking more about actually failing, not because were actually planning on doing so, but who did we affect and how did we affect people during that journey? Maybe it's three years. Do we do we match a lot of engineers to great companies? Awesome, I'm really happy that we did that. What else do we give?

We're a mission-driven company and that means that my dream honestly is seeing we already do this, we've started doing this, where every contract that we sign takes a kid off the streets in Kenya. I've taken some of these kids off the streets myself with my own hands and watched them to go from 10-years-old huffing glue, homeless to having a safe place to stay going to school, having parental figures in their lives.

If we sent a couple thousand kids to school and got them off the streets and failed after three years, I think that would be awesome, to be honest with you. We also want to get back to open source and we want to make this a company where grace is just central to the engine continuing to run. That might mean going above and beyond to help one of our engineers out, who doesn't necessarily deserve it, but that's not what grace is about. It's about a gift and loving our engineer as well.

Same thing with their internal culture; making sure that there's – humility is a big piece of that, the guys, though I'm the founder, they do performance reviews for me. I think that everything I've learned from marriage counseling, I've gotten to bring in as a founder to doing well interpersonally with my team.

It's some of the aspects that I just mentioned that that's what I really care about, and I think as a byproduct we'll be able to grow healthy as a company and larger, but as a mission-driven company it's not just all about getting to the 100 million dollar mark, or even just how to do it. It's about what really matters to you and your team.

[0:58:46.9] CS: I think to that point to, as we look to build more internal tools and become more efficient in some of our processes, there's always I think a trade-off of when you automate a process, you also remove some of the human element for it. In some processes that's fine and that's what you want. Others, that human-elephant actually or the human elements, the defining factor of what makes us different, I think.

We place a very high value on the relationships we have both with our clients and with the developers who come through here. Even if for some reason the developer never gets a job through us, we want to treat them well and we want them to leave and say those guys did right by me as far as in their ability to do so.

There's definitely a balance of how we're talking, about how to be more efficient and implement things like automation, like machine learning down the road, all of these types of things we want to bring in, but also have the human side, the gray side that Gabe was talking about, where we're able to really know the people that we're working with, we're able to take care of them while we have a relationship with them, and we just continue to have good momentum in that way.

I mean, our business is a lot of referrals. I don't think that was necessarily designed but it's I think it's just a byproduct of Gabe setting the culture of how people are to be treated and that really being such a big part of what we do. Even companies who never hire an engineer for us will send us referrals, because they trust us and I think that just speaks a lot to the culture that's been built here.

I came in a year ago where Gabe had already been doing this for a while and it was infectious. It was already built-in, the habits were there and I think that's such a big deal for why we're able to grow and continue to see success.

[1:00:40.6] JM: All right, guys. Well, it's been awesome talking to you and I've enjoyed working with G2i in the past. Thanks for building a great business and thanks for coming on the show.

[1:00:49.6] LJ: Thank you.

[1:00:49.9] GG: Thanks for having us, Jeff. It was a blast.

[1:00:51.4] CS: Yeah, thank you Jeff.

[END OF INTERVIEW]

[1:00:56.7] JM: You're a successful developer and you couldn't have gotten to where you are without help in your education and career. Maybe you're thinking about ways to give back in the community where you live. The TEALS Program is looking for engineers from across the country to volunteer to teach computer science in high schools. Work with a computer science teacher in the classroom to bring development concepts to life through teamwork and determination.

Pay your success forward by volunteering with high school students in your area by encouraging them on the computer science path. You can make a difference. If you'd like to learn more about the Microsoft TEALS Program, or submit your volunteer application, go to tealsk12.org/sedaily. That's T-E-A-L-S-K-1-2.org/sedaily.

Thank you to the TEALS Program for being a sponsor of Software Engineering Daily.

[END]