# EPISODE 1019

[INTRODUCTION]

**[00:00:00] JM:** JavaScript fatigue; this phrase has been used to describe the confusion and exhaustion around the volume of different tools required to be productive as a JavaScript developer. Frameworks, package managers, typing systems, state management, GraphQL, deployment systems, there are so many decisions to make. In addition to the present-day tooling choices, a JavaScript developer needs to watch the emerging developments in the ecosystem. ReactJS is evolving at a rapid clip and newer primitives such as React Hooks and React Suspense allow developers to handle concurrency and networking more robustly.

Tejas Kumar works with G2i, a company that connects React developers with organizations that are looking for high-quality engineers. His role at G2i is head of vetting, which requires him to asses engineers for their competency and JavaScript related technologies. Tejas joins the show to discuss the modern stack of technologies that a React developer uses to build an application.

Full disclosure, G2i, which is where Tejas works, is a sponsor of Software Engineering Daily. Tejas is also speaking at Reactathon, a San Francisco JavaScript conference taking place March 30th and 31st in San Francisco. If you like this conversation, then you can hear from him at Reactathon. It's in San Francisco and it's got a lot of great JavaScript speakers. You can also hear more podcast episodes about React by listening to the Reactathon podcast, which is available at reactathon.com/podcast.

[SPONSOR MESSAGE]

**[00:01:45] JM:** If you can avoid it, you don't want to manage a database, and that's why MongoDB made MongoDB Atlas, a global cloud database service that runs on AWS, GCP and Azure. You can deploy a fully-managed MongoDB database in minutes with just a few clicks or API calls and MongoDB Atlas automates deployment and updates and scaling and more so that you can focus on your application instead of taking care of your database. You can get started for free at mongodb.com/atlas, and if you're already managing a MongoDB deployment, Atlas has a live migration service so that you can migrate your database easily and with minimal downtime, then get back to what matters. Stop managing your database and start using MongoDB Atlas. Go to mongodb.com/atlas.

[INTERVIEW]

**[00:02:45] JM:** Tejas Kumar, welcome to Software Engineering Daily.

**[00:02:47] TK:** Hey, happy to be here.

**[00:02:49] JM:** I want to start by getting some historical perspective over the last 6, 7 years of React. React came out and it as just a Vue layer, and we could still describe it that way, but it has had downstream impact on the rest of web development. What have been the downstream impacts of React?

**[00:03:12] TK:** Laurie Voss did a really great talk at JSConf EU last year where he kind of shared a lot of insights based on npm's data, and one statement he made was really profound. He said React has dominated the web, right? That's a downstream effect if I understand the question correctly. I think the biggest effect and really the most profound is that React brought this like simple yet extremely clever component model to the web, which was desperately missing at the time.

I mean, you think of JQuery prototypes, script.aculo.us, all these stuff that came before, and there was coupling and it was very hard to refactor like a large scale application. Even the backbone in marionette back in the day, and React just brought this component model that now people have adapted, even Angular and Vue. I think that honestly is the biggest effect that React has had in the web industry.

**[00:04:07] JM:** I think of the React ecosystem as it's a demarcation point in post-Rails web development, and that's not entirely fair description obviously because Rails is a fully-fledged framework for building a web application and React is just a frontend that sort of had all these downstream impacts, but the fact that it's open-ended, that has really let the web take a different direction than the Rails ecosystem. Do you have any perspective on how that open-ended nature of React rather than the out-of-the-box experience of Rails? How has that affected web development?

**[00:04:49] TK:** Well, I can tell you how it has affected me as a web developer and many, if not all of my peers, right? This is not just my peers who work with React, but my peers who work with Angular and other kind of – I don't want to call them frameworks, although Angular is a framework, but you get the idea. Is that the open-endedness has kind of been applauded by everyone as being this thing that cultivates community. Because of the open-endedness of React, we see things like React router. We see things like emotion, very popular CSS and JS library. Even I've made a few libraries just to solve some problems that are not React's to solve.

It kind of gives an opportunity to other developers and say, "Hey, this is an amazing project, an amazing ecosystem and I'd like to contribute to it, and I don't need to contribute to the core. I can still contribute by virtue of like a library or something."

I also think in keeping it open-ended, React has really modeled a fundamental principle of software engineering that I really appreciate, which is the single responsibility principle, because React, from the get go, was just meant to solve one problem and solve it really well. I don't think – I may be wrong here. Jordan Walke might correct me. You know what? I don't think it was intended to be created to solve all the problems like authentication, like routing. But no, it just solved one problem and solve it really well, and it does.

The other problems, it's pluggable and modular with things like hooks where we can chime in, but it does what it does and it does it well. I think that is something that has also influenced the way I along with my friend create software, is we usually will now create these units that do their job and their job well and then integrate them. Whereas in prior times, it was Wild West cowboy land.

**[00:06:43] JM:** Today there are React-centered frameworks. What role did the React-based frameworks serve?

**[00:06:51] TK:** I love this question, because my personal website and my blog is built using a React framework by some friends I really respect and appreciate. The people over at Zeit. My friend Guillermo and Tim Neutkens create NexJS and it’s this framework that really just solves a whole bunch of things that I know how to do, but I don’t really want to do them.

I don’t want to configure webpack. I don’t want to set up like a routing structure. I don’t want to set up server-side rendering. How cool would it be if I could just like create a new project? Create a pages folder and put my pages in there and then it magically becomes either a static website or some server-rendered thing based on the content. Well, that’s what the framework does. I think that’s amazing, because I could build – I actually started working on a website for my mother-in-law, and it’s just I create a new folder, pages, put some stuff in there and it’s just – I don’t even have to run like next dev even. I just type in next in the terminal and I spin up a local dev server. It’s so magically.

I think to answer your question, I think these frameworks really bring the magic and they do something that really React tries to do and does well and succeeds. React tries to abstract the DOM and web APIs away from people, so they write React components, and then React does the rest. React renders it to the DOM. React diffs and all that jazz. You can essentially focus on the product you want to build. You don’t have to think about DOM APIs. You don’t have to think about events. You don't have to think about updating certain parts of your app. You don’t have to think about optimizing them.

I think frameworks take that approach and follow it to say, “Hey, you can focus on building your product. You don’t need to think about dev tooling. You need to think about server rendering. You don't even need to make the decision if this should be static-rendered or server-rendered. All that's handled. You can just create your great thing that you want to create. I think that's what they solve.

**[00:08:52] JM:** Right. So you've touched on two sides of the development process that they help with, the getting started boilerplate side of things, as well as the scalability side of things, and I think the boilerplate side of things is if I was a React developer when React first came out, or I'm like a new web developer when React first came out, I hear that React is the thing to do, but I don't know enough about web development to really piece together what should my backend be. I mean, people tell me node, but I don't know. On the later stage side of things, React helps with scalability issues, which might be manifested in the server-side rendering question, like when am I rendering my pages?

Can you dive a little bit deeper into each of those things? So you have on the one side, the beginner, the I'm just starting a new web application boilerplate side of things, and then on the other end you have the I'm a later stage React application. I need to figure out how to scale my application. I need to think about server-side rendering. Take me through each of those sides of the development process and how the frameworks help with that.

**[00:10:02] TK:** Sure, yeah. I actually mentor a ton of people on Twitter, like we'll talk over DM's and we'll kind of figure out the best way to do things, or at least the best way that I can see that we can see together. If you're a beginner and you hear React to the thing, how do I do it? You pretty much go on dev.2 or medium and go, "Hey, how do I –" You search for blog posts and follow it, or actually this happened as well. A friend of mine just found like a GitHub repo that was like a starter kit and just cloned it or use Create React App, which I don't know if Create React App qualifies as a framework, but it is a boilerplate, right?

They do that, you start is and then you can just create components. Then it's a matter of following the React docs, creating components and so on. I think the frameworks really compete in a sense, if you know what I mean. Would create React app, because frameworks do what Create React App does, but more.

I think in terms of the boilerplate stuff, I don't see that much value from frameworks. I think the bigger value comes from the later stage stuff, which even if you're a beginner and you've built something, you have a boilerplate and it's working and it starts to get traction and suddenly it's huge and you're seeing millions, billions of hits. How do we handle this scale? That's where the frameworks really shine, particularly NextJS, because when you couple it with – This isn't like a paid placement or anything. I just have a deep respect for these products. When you couple it with Now, which is the Zeit's cloud solution, literally, you don't have to care about scale. I don't have to care – My website is built this way. I don't care about scale at all.

When I hit thousands, when I hit millions, like it just horizontally scales. For the static stuff, it uses CDN and places static assets on edge nodes. For the server-side stuff, I don't even know what it does really, but I trust it and it works. All that to say, the frameworks, they take this complexity away from me so I'm going to just focus on creating content that my friends appreciate.

**[00:12:08] JM:** Let's talk a little bit more about the specific frameworks. We've been talking about frameworks in the abstract. The two that I know of that are the most popular are Gatsby and NextJS. Tell me a little bit more about how these two frameworks, what their market segmentation is, what problems they specifically are built to solve.

**[00:12:29] TK:** This is a really good question, because I was thinking about this recently. I was thinking about if I was to take something huge, like MDN. MDN has got like tens of thousands of articles on pretty much everything on the web. If I was to rebuild it from scratch – By the way, this is how I learned. If I was to rebuild it from scratch, what would I use? Would I use a framework? What framework would I use? Would I use Next? Would I use Gatsby? Would I even use some type of Vue framework?

While thinking it through, at scale, at like a huge wiki-sized type situation like MDN, I think I wouldn't use Gatsby in this case. The reason for that is because I don't think Gatsby was created to solve that problem. Now, I could be wrong and somebody will tweet at me probably saying I'm wrong, but from my experience to my best knowledge, Gatsby doesn't seem well-suited to that solution because it is a static site renderer. Meaning it will traverse your files, your articles and with linear at best complexity one by one will output 1-to-1 HTML file, like static files that go on the cloud. That's great for like small to medium-sized sized websites, and that's I think what Gatsby is better suited to.

Whereas for a problem of this nature, tens of thousands, maybe even hundreds of thousands of posts, I'd probably choose a different framework that is NextJS because it intelligently can tell the difference between static and server, and I don't even need to make this decision with Next. Based on my code, it knows, and at build time will decide to either build static assets or build out a server that then serves content.

For example, I have a friend, Monica Lent used to work at SumUp. That's right, SumUp, and she was a frontend tech lead there and she often dealt with scale. At scale, they had a static site system that would take hours and hours and hours to build because it had a lot of content. For that reason, I think it's faster to have an API, pull one article from a server and serve it to the client. So server rendering in that case. All that to say, concisely, Gatsby is great for static sites small to medium-size and NextJS for things that tend to have more content seems to be where I lend on this topic.

**[00:14:58] JM:** I want to talk through more tooling in the stack of React-related software. The bundler, things like webpack. What is the role of a bundler in a modern React application?

**[00:15:15] TK:** It's to bundle. For an internal tool, we're building at G2i. I actually had to build a bundler myself, which is cool, because I got to learn how bundlers work. I think in the context of modern React and in outside of the context of modern React, the bundler is pretty much what it says it is. It's something that you give it entry points. In the case of webpack, you write your webpack config and you say, "This is the entry point." Then what it will do is it will traverse, it will look at the imports of this thing and say, "Okay, he or she or it needs this file, this file, this file," and will create what it calls a dependency graph. Ultimately combining the results of each graph in the right order in one file, so effectively bundling all of your dependencies in one place.

The real challenge here is figuring out what depends on what and then bundling them in the right order, because if something is not defined when you need it, then your app crashes, right? Essentially, that is what a bundler is and that's what a bundler has been. To this day, with React or without, that's what a bundler is.

**[00:16:23] JM:** Is there a reason to use a bundler other than webpack?

**[00:16:27] TK:** I think the web is rich and diverse, and I don't just mean rich with money, mind you. It is diverse and there are tons of use cases and things. So absolutely, even if I don't see it yet, I'd say absolutely. Now with Create React App, you just get webpack out-of-the-box. By default, most React apps would have webpack, but I have some friends over on the chrome team who put together a project with Preact and deliberately chose not to use webpack because it has certain limitations with modern EcmaScript modules, or so I've heard. In this case, they opted to use Rollup.

But personally, what I'm really excited about is a project called Snowpack, which I don't know if you've heard of Snowpack. It is an incredible project. It aims to use – It creates a folder for web modules that are just plain old modern JavaScript ES modules and uses those instead of like node modules and common JS style. Meaning you can lazy load and tree shake and actually ship modern real JavaScript to your users. It's kind of scary to think about how much nonstandard JavaScript worshiping to our users at scale, but Snowpack is solving that problem, so it's very exciting to watch.

**[00:17:50] JM:** When you say nonstandard JavaScript, what do you mean?

**[00:17:54] TK:** One example I can give you is like imports, import statements, local imports, like import function from and then you specify a relative path like "./myfilename". That is nonstandard JavaScript. This is to the best of my knowledge. Not on the JavaScript specification, but these local imports are a NodeJS concept. In the world of web JavaScript that you shipped to a user, it's extremely unlikely, if not impossible to have local imports. You would always have absolute paths, even specifying a protocol with modern real – Rather standard JavaScript imports would have absolute important paths. That's one example.

We've grown accustomed to like this hybrid NodeJS style situation, which is nonstandard JavaScrip. I think we'd all be a little bit better off using modern JavaScript. For me personally as a developer, it doesn't really make a difference one way or another, but from the user's perspective, I think they would be served better.

**[00:19:02] JM:** Why is Typescript used in React applications?

**[00:19:05] TK:** Typescript is used in React applications in order to make re-factoring a little bit more comfortable and to really help them scale better. I've worked on web applications for years for very, very long time, and until I started using Typescript professionally, I had literal nightmares at my job about deleting code or about trying to re-factor something, because I had no idea what it would break. Oftentimes, things would break. I needed a safety.

So Typescript, like before the project even builds, will tell you, "Hey, just so you know, this thing that you re-factored broke this other thing that you weren't even thinking about." That code will not compile and I will not ship it to my users. So Typescript really saves me as an engineer and also saves the product or company that I'm working for.

**[00:20:02] JM:** That's because of basically type safety together with a compile time step?

**[00:20:10] TK:** Yeah. Typescript is a superset of JavaScript. It's very much like JavaScript except it has just a little bit of extra syntax, and the other thing is it's compiled ahead of time. JavaScript is compiled in time. If you go on a website, you download JavaScript, you execute it, that's when your JavaScript is compiled on the fly. Typescript is compiled when you run like Yarn build or npm run build. It is like compiling Typescript is your build step, which will fail if you're doing something illogical in your code that you don't yet see.

For example, if you have a comparison, like an if statement, and if you're comparing effectively if true -=== false. If you're doing a comparison that will always be false or always be true, but it might not look like it to you because you're using variable names and you can't decipher them, Typescript will tell you. It will literally straight up tell you, “Hey, this is illogical. So consider deleting it.” It'll tell you that before your project builds using static type analysis. Effectively, Typescript sits on your computer and literally it just watches your code as if it was some type of code stalker.

[SPONSOR MESSAGE]

**[00:21:31] JM:** Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we’re being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it’s definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

**[00:23:07] JM:** We discussed server-side rendering a little bit earlier. This is something I'd like to get in deeper with you. Applications can be rendered on the server or rendered on the client. In some cases, you're doing a combination of the two. How should a developer weigh the options of client-side versus server-side rendering?

**[00:23:29] TK:** This question hits really close to home, because I recently had a law of conversations with people criticizing me on the Internet for using GitHub as my backend for my blog and server-side rendering everything instead of statically building. The argument was always, "Hey, well a static, you can put it on the CDN." Then I said, "Great, that's amazing that you can do that with static." I made the decision for server-side rendering because I actually compared both of them like static versus server-side and my key decision maker was the time it would take to get it to my users. I believe that should be everybody's key decision-maker, is can my users be happy with this or not?

If it's going to take like 30 seconds more to get something to my users with server-side rendering, theni would choose static without even thinking about it. In my case, it was a deviation of like 3 to 7 milliseconds, which doesn't really mean that much, honestly, to me. I was really comfortable choosing server rendering because I was like this is before someone even blinks. But I think, yeah, that's how I would choose typically between server or static is how fast can I get stuff to users. But also of course the other factor is how much stuff do I have to build?

The other reason I chose to server-render my blog, for example, is because I foresee it – I tend to talk a lot. I don't know if you've noticed in the course of this interview or in the course of this podcast, but I foresee my blog crossing 100, 200, 1,000 articles over the course of its lifetime, and if a static site generator has to build each of those into HTML files one at a time, I don't want to sit and wait for that, which is why this is the other reason I chose server rendering. I think those two for me are the biggest factors.

**[00:25:23] JM:** It's worth pointing out that today we have better tooling for doing this. I mean, you go back, whatever, 5, 10 years to Heroku or deploying stuff directly to AWS, Elastic Beanstalk, whatever, you didn't really have a whole lot of options. I think most the time you were just shipping your JavaScript framework, like backbone, to – I could be wrong, but you're shipping like backbone to the developer or to the user and the user's browser has to parse all the necessary JavaScript to have the framework load and then have the framework interpret the custom framework code on their browser, which just contributes to significant load time.

Today we have tools that can do this on the server side and we have cloud providers that even tightly integrate this server-side rendering into the way the you develop your application and you have commonly referred to a static site hosting. You can correct me if I'm wrong. I think you've done more web development than I have. So maybe you can give us the state of the hosting providers and the tooling for doing server-side rendering. Explain why this is more of a conversation point today.

**[00:26:46] TK:** Yeah, I think it's more of a conversation point today because of the – I call it the serverless movement. Everybody is talking about serverless. Everybody is talking about JAMStack, and it's really interesting to see how really serverless or JAMStack solves kind of the same problem that React does.

What does that mean? I mean, the problem that React solves is you focus on building your product, not the DOM APIs, right? Similarly, serverless, in this case, static site hosting, not functions, but static site hosting, solves the problem of you focus on building your product or service and not where it lives or how it scales or how it's served to your users.

In that way, I think that's why it's a talking point. It's this thing that facilitates rapid development and can be monetized very well. For example, if I was to start a company, I would just go on like AWS or Zeit or Netlify or something or Heroku. Probably not Heroku, because it's serverless. I don’t have a server. I would just sign up, tell it were my GitHub repos is, and “Bam!” it's in the cloud.

Now, this isn't a problem for me because it's all free in the beginning, and when I start to actually see some traction, then I pay them. They can charge whatever they want within reason. I think they monetize it and they effectively run their businesses and I get my product in the cloud. I think that's the main reason why this is such a talking point.

It doesn't really solve the problem of shipping like a lot of code and then a framework understanding. That still happens. For example, React will still download on to your browser and then the React app will still download onto your browser and talk to React and so on. But the problem space has gotten way simpler because we don't need to think about servers anymore.

I will say though that it is also faster because the speed in which if we were even shipping backbone today, the speed in which it downloads could be significantly faster because these providers, these cloud providers would store these assets on something called a CDN node or a content delivery network where they can cash them and just serve them significantly faster than if they were to load fresh every time. That's the benefit. But ultimately, it's all solving the same problem of make things simpler for developers and get stuff to users faster.

**[00:29:10] JM:** You did point out the increased role of the CDN in today's web development. The idea that we’re rendering these applications on the server side and then pushing them out to the CDN and then the user is accessing them directly from a CDN. Does this introduce any complexity in how we’re creating our applications? Because if you have this additional node that is participating in the development of your application, it seems like it's harder to have an entirely consistent experience for the end-user, because if I write a comment to a blog and then that blog needs to – That comment needs to be posted to the blogs backend, then the blog needs to do some server-side rendering, then the blog needs to serve that server-side rendered application to the CDN and then the CDN is what serves it to the other user that would be reading my blog. Does that create any consistency issues?

**[00:30:16] TK:** It does, and I don't really know what the solution is to that to be honest. I have been looking at a database. I’ve been looking at FaunaDB because one of their big marketing points is immediate consistency, right? So if you were to talk to their DB via an API, it would be faster. But yeah, for sure, there could be consistency issues there.

**[00:30:40] JM:** Wait. What would that even do for you? I mean, that’s just a database consistency. I mean, we’re talking about consistency between a CDN and the actual cloud provider server that's doing the rendering.

**[00:30:55] TK:** Correct. This is a problem space that I actually haven't worked in yet. It's defi nitely interesting. I know Phil Hawksworth at Netlify did a talk about this at a conference in Romania last year. The video is on YouTube and I did watch it live. I’ve been meaning to go watch it again, but this stuff really interests me and I'd love to understand it better.

**[]00:31:15 JM:** Me too. One thing I do understand is like – I did a show about FaunaDB and it seems like a great consistent database. What I don’t understand is why the JAMStack people like it so much. I mean, do you know anything about what does it do outside of like what you would get out of the – Whatever. DynamoDB? Whatever other database I would take off the shelf.

**[00:31:36] TK:** Yeah, that's a really good question. Funny thing, I've been kind of just playing with Fauna. So I don't have it in production, but I've been playing with it, and it's apparently based on this paper that talks about modeling data for scale. That's what I'd like to understand more. It looks really academic and interesting. But for me as a JAMStack person, the value I see in Fauna is the developer experience. Like I create a database, I populate it and, “Bam!” Just like that, there's a GraphQL API that I can connect my JAMStack app to. I think that's why JAMStack people like it so much, it’s because it's simple. It's kind of like the same reason JAMStack people like Netlify so much. This I know for sure, is because when you go in netlify.com, you click a link, connect to GitHub, set a repo and then you’re live. It's that simplicity that I think the JAMStack people tend to appreciate. It's also similar to why I think Hasura is so popular, it's because without any work, you point to like a Postgres database on like Amazon RDS and you have a GraphQL API. I think that's what we tend to appreciate.

**[00:32:44] JM:** Rights. It’s probably some developer experience. Some part of the developer experience that I wouldn't understand if I didn't build an application with this thing.

**[00:32:52] TK:** Yeah. I was playing with Fauna Shell and, literally, you just type create. You create a table, you create blog posts, and just like that, they're available via GraphQL API. I've done absolutely nothing except write a blog post inside of like a black function call and it just is there. I think that's the value.

**[00:33:12] JM:** You mentioned Hasura. Hasura is a GraphQL on database tool. I think it helps you design your GraphQL schema such that you can talk in GraphQL to a Postgres database, or that was at least like the first use case. They branched off in other things. Tell me about GraphQL usage in the React ecosystem today and the state of GraphQL tooling.

**[00:33:40] TK:** I see a lot more GraphQL usage in the React ecosystem today than I did just a few months ago, and I haven't actually looked at the state of JS Survey, but I'm sure I would be very surprised if it said something different. GraphQL is here to stay.

I did a talk about this recently. I think the reason it's here to stay and the reason it's growing in popularity is because it is nothing more than a specification. It's kind of like you asked me about the open-endedness of React earlier in this call. It's kind of like that. GraphQL is just an open-ended document and it says, "You implement it however you want to." People like Apollo, Hasura, even Prisma took some liberties and were like, "Okay, great. We have a specification. Let's go. Let's create products."

With that, we have some amazing tooling available in the React ecosystem. We have – To this day, I use Apollo client in production and it is a dream. I wrote a subscription today and it's just real-time, just like that. Speaking similarly of tooling, with Hasura, personally, is a dream come true, because – It has actually nothing to do with GraphQL besides the API, which I'd really love. With Hasura, it's more about Postgres than GraphQL. You configure it, you point it at Postgres database. It can live anywhere in the cloud, locally, whatever. Just like that, literally, magically, it all maps to a GraphQL endpoint, which you can then query. You can choose what in your database you want to track and what you don't want to track. You can expose only certain tables to your GraphQL API, and that is just pretty bananas if you think about it. That is just that simple to set up.

More than that, it also works out-of-the-box with GraphQL subscriptions. You could create real-time data, like I showed this to my company, G2i, today, because we work with developer profiles and so on. When somebody gets a job, we want to know. We have numbers of how many people are currently occupied with clients and how many people are currently free, ready for higher, and these numbers are in flux often because people are getting hired or people are completing projects. We literally have this now real-time. You don't need to reload your browser window. Just like you keep the window open and the server will push you new numbers using GraphQL subscriptions, and Hasura just magically wired that up for free. We didn't have to do anything for it. All that to say, I'm really excited about the tools in this space, like specifically Hasura on GraphQL, on Postgres.

**[00:36:17] JM:** Yeah. What you point out there with the subscriptions thing, one way of looking at GraphQL, the way that I saw it presented when it first came out was here is this piece of middleware where you can describe your needs to the GraphQL middleware, and the GraphQL server will go out and query the databases that you have. Maybe you have like five or 10 different databases if your Facebook, or you're Netflix, or you're Airbnb. You have all these different data sources and it's very complex to query these things. So GraphQL stands in and federates the queries to the different data sources you have.

But with subscriptions, you're describing a use case of GraphQL that's useful even if you only have one database. Just the idea that you have this layer over your database that fulfills the subscription API. Basically, the idea that your application can be monitoring the changes in your database. This seems like something that will be useful even if you only have one database. It's kind of like and an improvement to the idea of just using GraphQL for Federation.

**[00:37:35] TK:** Definitely, because GraphQL is an API specification. Surprise, surprise. There isn't one at all on the web. Maybe, maybe there is you could say Swagger open API is, but I'm not entirely sure. For example, what do I mean by that? I mean, if I have a database, just a raw database, nothing on it, and I want to mess with it. I want to read. I want to update records. I want to subscribe maybe to event triggers. How do I do it? There is not a formal document saying what an API should look like except the GraphQL document, right? It even handles subscriptions, which Postgres databases support event triggers so that connection, that conceptual connection is extremely compatible, and that's exactly what Hasura does.

Actually, right now we use – Because at G2i, we're not Airbnb size yet. We just Hasura on one database mainly for the subscriptions and the API that we just get out-of-the-box. That way we don't have to sit and create an API. It just exists.

**[00:38:41] JM:** I'd like to talk more about modern React development and just get an overview for some of the recent changes in your perspective on them. One is the context API, and I know that the original most popular system for state management was Redus, and I understand that Context has supplanted some of the use cases of Redux. Can you explain what the context API does differently than Redux, and maybe just give an overview for the problem of modern state management in React applications?

**[00:39:17] TK:** Sure. Yeah. I'm not entirely sure that the Context API has supplanted Redux, mainly because Redux itself used the – Like React, Redus used the Context API to provide a root level store to a React app. The Context API was –

**[00:39:32] JM:** Oh! So am I just totally confused about Context being something newer?

**[00:39:36] TK:** Possibly. Just in that React has had Context for long time. Though the API – You're right. The API was often in Flux. It was changing until recently. But I think what you're talking about, what you might mean is that useReducer has supplanted Redux. That is a statement I would get behind.

With React Hooks, we have this new Hook, it's called useReducer, and it works by accepting a reducer as input and an initial state, and then you dispatch actions through a dispatch. It's very similar to Redux, but it's like baked into React. It's a first-class citizen. That, for sure, like in my project has 100% supplanted Redux.

I was a tech lead at my previous job at a company called Contiamo, and one of, really, my best friends, his name is Fabio. He wanted to use Redux in a project and I just basically vetoed that and I said, "No. No. No. We got a use useReducer," because we're shipping less code that way. We're not bundling. We talked about bundlers not bundling Redux. We're not bundling something like Redux observable for middleware, and we're just shipping React only.

It's a big win, because we ship less code, and that for me is the biggest win. But also, React had this concept of middlewares. If you've worked with Redux, you have used something like Redux Thunk, Redux Observable, Redux Saga, and these things existed to allow for like stateful side effects inside what is supposed to be a pure store. How do you do that using middlewares?

Now the problem with this is there were many of them and you had to kind of know some of them to fit into a team that has been using them. So there's already like a barrier to joining a team if you don't know the middleware. Secondly, some of them just end up unmaintained or they're hard to – There're tradeoffs between them is what I'm trying to say.

With useReducer and React, you don't have to deal with any of that because with the new useReducer hook, React also exposes a useEffect hook, which is effectively what Redux middleware would do. React now ships with useState, useReducer, useContext, useMemo, useCallback, and these hooks really interplead very nicely to make, in my projects at least, Redux not necessarily useful.

[SPONSOR MESSAGE]

**[00:42:04] JM:** DigitalOcean makes infrastructure simple. I continue to use DigitalOcean because of the low friction and attention to user experience. DigitalOcean has kept the experience simple and I can spin up a server in less than a minute and get high quality performance for a low price. For an application that needs to scale, DigitalOcean has CPU optimized droplets, memory optimized droplets, managed databases, managed Kubernetes and many more products. DigitalOcean has the flexibility to choose the right instance for the right workload and he could mix-and-match different configurations of CPU and RAM.

If you get stuck, DigitalOcean has thousands of high-quality tutorials, responsive Q&A forums and a customer team who treats customers respectfully. DigitalOcean lets developers focus on what they are building. Visit do.co/sedaily and receive $100 in credit over 60 days. That $100 can be put towards hosting or infrastructure and that includes managed databases, a managed Kubernetes service and more. If you want to get started with Kubernetes, DigitalOcean is a great place to go. You can use your $100 to start building your distributed system and you can get that $100 in credit for free at do.co/sedaily.

Thank you to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

**[00:43:40] JM:** Right. So you've definitely exposed my lack of detailed understanding of this ecosystem. I wanted to do like a week of shows on React and JavaScript, and I've been doing a lot of coverage and I think I have – I've been touching on elements of the React ecosystem that people care about and hopefully asking the right questions. But definitely as an outside observer, there are a number of things that I simply do not understand very well, but that doesn't stop me from asking about them.

**[00:44:09] TK:** Yeah, I'm really glad you asked, because it helps me understand them too.

**[00:44:13] JM:** Right. Getting further into things that I don't understand very well, but people want to hear about. There are some other new primitives that have gotten introduced into React more recently, so Hooks and Suspense. Why does React keep introducing these newer primitives for React development?

**[00:44:37] TK:** Short answer, because Facebook needs them. Facebook, it's a huge app and there are things that they need to do to make the user experience better, and that's what I want to touch on, is React fundamentally exists to improve end-user experience through improving developer experience.

There's a sweet spot where the pleasantries of DX and UX, that is developer experience and user experience, meet, and that's effectively why React exists. My opinion and belief is to meet at that point and provide pleasant experiences to developers, which it certainly does for me and pleasant experiences to users through the use of just elegant UIs.

Now, you asked about Hooks and Suspense. Hooks exist, they do solve a problems. The problems that hooks solve for developers is that they help us kind of group are stateful logics.

React components in and of themselves ought to be stateless and free from side effects, so stateless and pure. But that's obviously not going to work if you have an app. You need state, such as user input or something.

So hooks kind of help you group your stateful logic. They're just expressive primitive wherein you can say, "Hey, this is what my state is. these are the side effects I'm doing and this is something I want to memoize," and you very declaratively state these things and you can co-locate them with your render logic so that your code is easier to read and work with. Ergonomically, it's very comfortable for the developer.

For the user, it's kind of unbelievable how monumental the value that hooks bring, because the prior art to hooks were these class components with JavaScript classes that were very big and did not minify properly, because classes really have a lot of interconnected things and identifiers and so on that you can't easily mangle if you're minifying them. Also, you can't really optimize classes as well as you can functions in JavaScript.

Because hooks are entirely functional, like they're able to become so small when you minimize them before you ship them to your users, meaning users download less code, get things faster and are happier. That's the value of hooks. Suspense is also very similar. Suspense actually hasn't shipped like in its full version yet. That's something we're all very eagerly awaiting in a future version of React where something called Concurrent mode will be enabled.

The use case for this – I mean, the developer experience is another – In this case, the developer experience allows us to very declaratively state, "Hey, even if this part of my app isn't ready, show the rest of it and then show a spinner here," or we could just as easily say, "Don't show any spinners. Wait for everything to load and then finally show this to the user." We have more granular control over what to show to the user when.

If you go on twitter.com right now, you will just see like spinner after spinner after spinner. That may not be very nice. That allows developers to more declaratively specify what to wait for and what to not wait for. On the user side, it's also a pleasant experience, because you don't really have to look at spinners anymore. Your page kind of loads according to where your eyes meet.

You could, as a developer kind of say, “Load things from the top left to the bottom right.” So the users would get that benefit of things loading where they expect them to be.

All that to say, I think React keeps introducing these new primitives, yes, because Facebook needs them, but also because we need them. Me as a developer, it helps me make greater apps that are high quality and pleasing to my users, but also really fun to work with from my dev team.

**[00:48:38] JM:** You’ve mentioned that suspense is this system for helping with asynchronous data loading, but that means it’s going to help with user experiences that have variable network connections, for example.

**[00:48:54] TK:** Yes.

**[00:48:55] JM:** But he also said that there's not really a best practice for how to use it today. Is that true? If I’m a developer, how can I use suspense today?

**[00:49:05] TK:** That's a great question. I said it wasn't released in its full. This isn't even in its final form is what I'm trying to say. It is half released, and the half that's released you can use in production. We might be using it in production at G2. Suspense for code splitting is the part of Suspense that is released in React and ready to be used. How that works is you would – For example, if you have a website with many different pages, before Suspense for code splitting what would happen is you would kind of just – At least in my case, I just didn't care about splitting my code. When somebody comes on tejaskumar.com, they don't just load my initial index page. They load all the pages even though they don't need to. They just spend that bandwidth, lose that time.

Suspense for code splitting encourages the best practice of actually splitting your code. Lazy loading or loading on demand the pages when they’re requested. How Suspense does this is you use a helper from React. Let’s call it React lazy, and you import a page using React lazy. That isn't really imported until it's needed, and then you wrap your components in this like suspense component that is just exported from React and you give it a fallback. You say, “Hey, in case this isn't loaded yet, show this,” and that could be a spinner. It could be a catgif. It could be whatever you want. Then when your page finally loaded after the user has clicked it and requested it, then it serves the user what they want to see. All that to meet the goal of shipping way less code to the user. Just imagine, instead of downloading every route on initial load, you just download one.

**[00:50:49] JM:** We've been talking about the different facets of the React ecosystem and how to build a React application today. One way to summarize how all these things fit together is the JAMStack. It's a useful way to think about applications, I think. But why has the JAMStack changed the way that we do software deployments? I feel like the JAMStack has coincided with the rise of some newer hosting providers, like Netlify, and Zeit. Why is that? Why do users need a hosting platform that is centered around the JAMStack?

**[00:51:29] TK:** I think that JAMStack is so popular for the same reason that JavaScript is so popular, and that reason is it's got virtually no barrier to entry. I wanted to start with JavaScript, I open Chrome, I open the console and I write code. The JAMStack allows you to do that. You write a JavaScript – If you know React, you write a React app using some type of API. You don't need to write the API. You need to write the database. It's some external interface. The last part is you have M in the JAMStack markup, which is static-generated content. You write JavaScript, talk to something with the API that generates markup for you, and this markup is static. Can be served from edge nodes on a CDN and it's this beautiful intersection of approachable, but also incredibly powerful when used at scale. For example, smashing magazine, right? That's a JAMstack site. I think that's the draw. I think that's why so many hosting providers have sprung up because it's something with a lot of volume because it's so approachable.

If there were no hosting providers, you'd build JAMstack sites, because why not? They're easy, and then you'd be like, "Oh, okay. What do I do with this now? Where do I put it? Do I start up my own node server or do I go on Heroku and kind of start – What do I do?" and these companies cite Netlify, show up and say, "Hey, give us your JAMstack so that we'll make it happen. We'll will even give you like TLS encryption and everything." I think that's why. It's like there's this whole bunch of people creating on JAMstack with nowhere to put it. That's the problem they solve.

**[00:53:09] JM:** What are the biggest gaps in tooling in web development today?

**[00:53:13] TK:** I thought about this question a lot. I genuinely don't know. If there is one, I'm sure it will be filled soon. I hear a lot of people say, “Hey, I'm sure there's an npm package for that.” That's kind of the frontend or JavaScript developer joke, is there is an npm package for that.

A friend of mine had no Internet in his house, like someone had cut the cable. So what he did in the – Like he was using office Wi-Fi. He typed in npm install Internet as a joke, because he had no Internet. It turns out there's an npm package for that. Gaps and tools –

**[00:53:45] JM:** What does it do?

**[00:53:47] TK:** I have no idea. I don't think. I think it’s maybe every website that has ever been made. Probably not. I don't know. But I don't know what the gaps and tooling are, man. Really, I’d give it a lot of thought. No idea.

**[00:53:59] JM:** Internet, I’m looking at the npm package. P is a small framework used to create browser to browser networks.

**[00:54:10] TK:** Wow!

**[00:54:11] JM:** After a connection is established, the middleman is no longer necessary. No proxies are involved. Okay. Whatever. Interesting. I'm glad this guy domain squatted on Internet for npm with his peer to peer networking package. It looks cool.

**[00:54:29] TK:** If you have a shell up and running, if you type in NPX Tejas, as in my name, you’d see something funny. All that to say, there’s just packets for everything.

**[00:54:37] JM:** There are packets for everything. You are a part of G2i, which is a company that has React contracting and works with large companies that need – Well, small companies too, that need React developers. If I am a new React developer today and I'm trying to understand what are the skills I need or what's the basic skill level I need to be at in order to have marketable React talent, how do I know that I have the sufficient skills?

**[00:55:08] TK:** Personally, for me, how I often find that if I have the skills or not is to interview for jobs. At G2i, that's kind of our business model is. What we do is we vet. If you go on our website, it says we vet, you hire. It's that simple. That's kind of what we do, is we look at applications from people and tell them, “Hey, your skills are marketable and there is like these companies that actually want you right now,” or we say, “Hey, there are companies out there looking for these things,” and those aren't necessarily your skills. It might help you to learn those things. That’s kind of what we do, is we help companies find the right people, but also we help people find the right skills for the companies, if that makes sense.

**[00:55:51] JM:** Definitely. But more specific level, is there like some minimum – You're talking about like, “Okay, the skills I need to interview and be successful.” Is there some subset of the skills that a superb React developer would have? What do I need to know to be a successful enough developer to get a job?

**[00:56:13] TK:** I think to get a job, you have to have an understanding of JavaScript. That's absolutely essential, and basic React. Honestly, like I've seen people get jobs who have learned JavaScript. There’s a book, it's called *You Don't Know JavaScript*. Great, great book. So they'll read that or do a boot camp, learn JavaScript, and then read the React documentation. Honestly, top to bottom, the entire documentation that doesn't – I have done it a few times over. It doesn't take very long. Literally, just based on that, people have gotten jobs. That's kind of what we're seeing today. But then of course like that's more for just an entry-level or intermediate React position. The more senior positions require knowledge of scale and thinking about what to memoize versus what not to memorize. Thinking about how can I reuse this? Thinking about not probably filling, but mocking things so that they can be automatically tested. Also, there's a whole another topic of knowing what to test, what not to test, what to type and so on.

I think those more nuanced topics tend to be what senior engineers do more. But by and large, it doesn't seem – If somebody knows JavaScript and has read the React docs and his built some type of pet project with React, it's pretty much a shoe and that they'll find a job.

**[00:57:35] JM:** To close off, what you do at G2i is you help vet people who are coming in. People who are applying to be a developer at the company, they go through this application process where you essentially screen them. What does that actually mean? What does a vetting process entail?

**[00:57:56] TK:** That's so controversial. Every time somebody asks me what I do with G2i, and I say I am the head of vetting or I lead the vetting team. I get like the stink eye, like, “Oh my gosh! Who do you think you are to vet, to evaluate, to judge? Are you a judge?” That's all I get. It’s all hate. Everybody hates me.

No. It's not really as controversial as it sounds. Yes, I know it's classic, but I would say this is a corrupt politician. But effectively what we’re doing is we’re connecting people with companies. A big concern in our vetting is bias, right? You think, “Oh, this person is a bad developer because they’re from some underrepresented community in tech or whatever.” Fortunately, there is no bias that way. Our bias is the requirements of our customers.

So we see our customers saying, “Hey I want this. I want this. I want this,” and that's kind of what we look for. Moreover, we automate a lot of that. We avoid like human biases. What the customers really usually want is literally just the React docs.

For example, the React docs will say, “This is a good practice. Do this,” and that is lifting state up, for example. Passing state down through props for components that share state. The React docs will also say, “Hey, don't do this.” For example, mutating state directly as supposed to calling useState, of for example, in the case where state maybe asynchronous, using the callbacks that are for set state.

The React docs has some strong opinions that we enforce in our vetting as well. But by and large, that's kind of it, it’s just like our customers want this. This is what the React docs says, and we cross-reference an application’s code sample that. The React docs does have some really strong opinions, and that's essentially what we do, is we follow the React docs and the requirements of our customers and we will cross-reference an application from a user from, from a developer, from a contractor. We’ll cross-reference their code sample with that, and that's essentially it. That's our vetting process.

**[01:00:08] JM:** Tejas, thanks for coming on the show. It's been really great talking to you.

**[01:00:11] TK:** Thanks very much.

[END OF INTERVIEW]

**[01:00:21] JM:** You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[END]