[INTRODUCTION]

[00:00:00] JM: Retool is a company that allows customers to build complex internal tools using a high-level GUI. Users configure the relationships between these different tools, giving them the ability to build applications even without much background in engineering. Of course, having engineering expertise helps as users can write JavaScript to interact with the higher-level components.

Snir Kodesh is the Head of Engineering at Retool. And he previously worked at Lyft as a Director of Engineering. He joins the show to talk about his work at Retool and how it compares to Lyft engineering as well as the lower level challenges of building the platform.

If you're interested in sponsoring Software Engineering Daily, reach out to us at sponsor@softwareengineering daily.com. We'd love to hear from you and bring your message to our audience. We reach over 250,000 developers per month.

[INTERVIEW]

[00:00:49] JM: Snir, welcome to the show.

[00:00:50] SK: Thanks for having me. Excited to be here.

[00:00:53] JM: Yeah, it's great to have you. So we've done a previous episode about Retool. And if listeners want to know some of the ground floor stuff on Retool, they can listen back to that episode. But I'd like to delve a little bit deeper into the engineering here. And I want to start by talking about your past experience at Lyft. And I just love to get from a high level how working at Lyft compares to Retool in terms of the engineering challenges.

[00:01:22] SK: Yeah, absolutely. I mean, I think, obviously, building an incredibly low latency marketplace and building a software engineering platform really couldn't be more different from sort of a product standpoint, and certainly, from an engineering standpoint. They do have a lot in common in the sense that we want both to be incredibly performant. And the approach to engineering, obviously, is going to involve high-performance and low latency systems in both.

But I think for Lyft, what I worked on was a lot of very large graph solves. So trying to be – In particular, in my role, which was in the marketplace function, trying to clear the marketplace, clear demand and clear supply with or the lowest latency and do that in a globally optimal way. So I'd say sort of one discrete difference there if you could probably infer is that a lot of ML models were involved in sort of the work that I did on Lyft to ultimately create an objective function push toward it.

Whereas with Retool, we are fundamentally a platform. So we are sort of unopinionated and don't have nearly as much bias about what the objective function should be other than to make engineers more productive and accelerate development. Well, we're not trying to shove you in one dimension or one class of applications versus another. We're not trying to be overly predictive in what you're trying to do. Because at the end of the day, we don't want to make false inferences. And we don't have the same sort of profit maximizing function that Lyft might have.

So very different engineering challenges in the sense that one is deliberately intended to be open-ended, in the case of Retool, whereas one is very opinionated from a company objective, in the case of sort of the marketplace functions at Lyft.

[00:02:54] JM: And when you think about the data model for Retool and the data structures, the primitives and you compare that to the primitives of Lyft, in Lyft, I think of rides, and drivers, and ETAs, and maps. In Retool, I guess I have a little bit less of an idea of what the core abstractions would be. Maybe Windows, or components. What are the core abstractions of Retool?

[00:03:32] SK: Yeah, so you got it completely right. And I think a lot of what I was mentioning in terms of constraints and opinions reflects in the data model. Where, exactly as you said, in the Lyft case, you have very sort of precise, specific objects that serve one particular function, right? Like a ride. Whereas with Retool, we have a notion of a page, which is much more abstract in terms of what sits within a page. We have components. We have queries. We have the metadata that associated within them. We have resources, which is the concept of where we're going and ultimately fetching data from. We have a bunch of audit events just to capture and

understand how you're using a page so that we can ultimately serve audit logs to our customers.

But each of these are really in some ways more of – Think of them, on some dimension, kind of like an interface that needs to be sort of instantiated by whatever a customer pushes onto them. Whereas you dif that against a much more sort of prescriptive schema in sort of the Lyft case. And of course, we take that to the limit, right? But part of what makes Retool really interesting at the end of the day is we embrace the concept of code within the confines of an application. So we want to give people just open-ended JavaScript that they can execute within, and obviously do that safely and do that performantly. But that's just another dimension of that sort of dynamism within a Retool application and within the Retool data model.

[00:04:52] JM: In the last episode, we talked about Retool pretty abstractly, as I mentioned. I'd like to talk about the architecture of a typical Retool app. When somebody makes an internal application, how that application is represented in-memory and on disk? And just how it's modeled?

[00:05:13] SK: Sure, yeah. So for starters, I think every application can fundamentally be represented as a graph, right, in terms of visually. Now that's not entirely how we store it on disk. I mean, I mentioned some of the data models that we construct, and there's obviously clear relationships between them that lend to that graph representation.

But as an example, you may have sort of a central query as part of your application that everything is dependent on. And that once that query runs a whole bunch of components go and sort of visualize the result of that query in the UI. And so we would effectively have component and component metadata that associates to that resource and subsequent that query and that query metadata. And so you hopefully – As I'm explaining it, it's clear how sort of that that graph and that dependency graph is built around the implications of the app itself.

So when you think about – And I've definitely listened to the last podcast. But just a little bit of a refresh, right? Fundamentally, what folks are doing when they come to Retool, they bring in their data sources, they write queries to extract information from those data sources. They connect those queries into components. Those components are then sort of organized into an app. And

fundamentally, what that means is that they may register event handlers on top of them that then control other components, right? You sort of have this almost workflow being created with an application, but you can represent that as a dependency graph.

And so we take our schema as sort of the more structured version of that, that maintains and stores all of its metadata. When we load it into the frontend, we're actually reconstructing that dependency graph for you in the application. And that's what ultimately allows the app to run.

[00:06:52] JM: Can you walk me through, I guess, the bootstrapping, the rendering of an internal app? Like how it gets pulled into memory?

[00:07:02] SK: In terms of how we – Sorry. Can I get a quick clarification on the question? Do you mean in terms of how we use, like, Webpack to bundle assets and load them, or something else?

[00:07:13] JM: Yeah. So like let's say I navigate to an internal applications page. And just walk me through what happens for that app to be loaded onto the page.

[00:07:26] SK: Sure. I mean, it's got several components to it. So for starters, we transform, as I mentioned, that schema into that graph, right? And we, again, bundle – We have all of our assets that map directly into those data models. We use Webpack to ultimately bundle them.

When we load them into the browser, I think there's a couple of steps that are, I think, particularly compelling, namely how we sort of take templatized JavaScript, run it, which is done through sort of a sandbox where we share complete app scope. And that's our way of effectively having safe JavaScript run in browser fully sort of hydrate styles, themes, templates, everything else, run them in the browser.

I'm not quite understanding the question around memory. So I might be sort of missing one key inside of what you're looking for. But I think that there is this really interesting component of Retool, which is people are writing active code. We are running it actively in the browser and sandbox.

[00:08:19] JM: Yeah, I guess the question was – Just when I think about the backend representation of the app, I guess these apps, they're just always around. I don't know why I was thinking that. I was just thinking – For some reason, I was thinking like the the app was structured in a file somewhere and then there **was** a separate database associated with it. But of course, now I understand it's just running all the time. You just have an active app that's running all the time.

[00:08:45] SK: To me, what's incredibly powerful, because as an editor, when you're building the application, you can fully interact with it, right? So it's almost this like real-time live programming environment. As you pull in a component, as you pull in a query, you press a button, you actually see the interactivity happen directly in the editor experience as opposed to publish that and then sort of load it separately **[inaudible 00:09:08]**.

[00:09:09] JM: So once I have a loaded application, can you describe how information flows between the components of that application?

[00:09:17] SK: Yeah, for sure. So, again, I think this is where that dependency graph really sort of takes hold and where a lot of its power is sort of exhibited. So fundamentally, like at the core, we have our queries, right? Like, I think, everything, for the most part, originates off the queries. I suppose you can build a static application that doesn't have an interesting data source associated with it, something like a static table. But most applications sort of originate with queries.

And I think, again, what's really interesting about our query structure is this is one of the dimensions in which we make sort of hard things simpler, right? So as an example, you can trigger actions off of a query success. That might mean you run another query. That might mean you mutate a component. That might mean you run some arbitrary code. But you basically get sort of guaranteed delivery of, "Hey, when this query succeeds, or alternatively, when it fails, we sort of commit and promise to take subsequent actions. And that's where we really just sort of like traverse the dependency graph and say, "Okay. Well, it's something. It's a state machine," right?

We say, "Hey, when this particular component completes its operation, assuming one forked for success, one forked for failure, take these sort of next subsequent events" But we just pass sort of the data directly through effectively global state. And you can mutate that global state. You can change it. But what's neat, again, is all components reference sort of just through variable naming other objects in the graph and are able to extract metadata directly from the APIs that those other objects expose. Does that sort of answer the questions? Does that help at all?

[00:10:45] JM: Yeah, yeah. absolutely. So maybe you could give me an example, like a chain of components in a typical application where data would flow between them.

[00:10:55] SK: Sure, yeah. So, yeah, I'm thinking of an app that I sort of built internally here. I feel comfortable sharing that and we really like to protect what our customers do for the most part. And I don't know how much I can share there. But I built this app that effectively runs our Q&A process for engineering, and to some degree, for the company. And so you can view it as a mix of, say, like a Slido with Reddit where people can show up and sort of upvote comments. You can sort of see that the state of questions that are in effect.

And so, firstly, you have sort of a submitting query. And unfortunately, I don't think we're recording video here. But I'd love to sort of demo it. As you submit a question, you automatically effectively take a post action. But that post action is represented in the form of just a query that writes to some table in some database. We then have a list view that is effectively constantly looking for state updates against that table. So as soon as that write query succeeds, we go and we refresh. We trigger a refresh on the list view table. That goes and runs a separate query against that table to retrieve data to publish and populate and effectively hydrate the list view.

So now you submit a question. You start to see it instantaneously generated because we're triggering the select query on successfully writing to the table, right? Writing to disk. And so you have one query that is a write on its success. We go ahead and repopulate the list view.

Same sort of thing for upvoting where you interact with the button, right? That button, sort of on it, we register an event handler on that button. That button then has a effectively a dependent step to running a query that populates the up vote. We have some sort of logic there for basic validation, ensuring that people can't sort of update multiple times over. And then that triggers,

again, sort of a write to disk, which of course, delightfully, as you probably can infer, triggers the list view to once more sort of refresh its local state.

And so you sort of – The query is ultimately the way in which data is flowing between these components. But the triggers for what is causing them to refresh, which is really sort of important from a performance standpoint, you don't want your list view constantly pulling your API or your data store directly is what really sort of makes it magical.

In this case of this application that I built, I wanted the scrappiest thing. So I'm just dealing directly with disk. You can imagine that for most people. There is some sort of front, right? Whether that's sort of a GRPC endpoint or some sort of RESTful interface to sort of help manage traffic, Etc. But most people don't want to just be writing directly to disk. But for me, internal use case is something that I want to do just in a really scrappy sort of way. And that's sort of how the data flows between the different parts of the application.

[00:13:28] JM: And with that application, can you give me a sense of some of the backing services, the, like, database services or infrastructure that underlies it?

[00:13:40] SK: Yeah, absolutely. I mean, in the case of my application, it's, again, pretty simplistic. I mean, it really is just Retool at a database, which on the one hand is really simple. On the other hand, it's quite powerful, right? I effectively was able to proxy for a full stack application just by having a database provisioned in a table that I could write to.

Now, again, I think for most folks, that's not the case. You want to sort of have that intermediate step of service that services basically. And that could really be anything. Again, I think for me, again, it's a Postgres table. But that could be any type of data store. And in the case of me wanting to insert some API for rate limiting, or security, or validation, or ACL checking and ensuring sort of auth and permissioning, that could be a Flask service on the backend. That could be a GRPC endpoint. Again, that could be really anything.

And I think if you look at sort of the set of services that Retool supports, it is quite expansive, including sort of services like Twilio, and Stripe, and Slack. And so there's a lot more that on a sort of any API that's really exposed, we can pull into Retool and make available.

So when I build an app in Retool, is it connecting to – Does it need to connect to a database that I have already defined or is already on my infrastructure? Or can I just ad hoc, spin up a database through Retool?

[00:15:06] SK: Yeah, that's a phenomenal question. So we're actually actively – We have a path that we've been working on for the last several months to actually provision and have sort of a fully managed solution. Best case scenario, it's super early stages. But your mind is going exactly to the right place.

I think for the most part, people have already sort of connected. And at sort of an organizational level, right, per enterprise per company, you sort of have the set of resources that are available to you. And then from an org management perspective, you can sort of allow certain groups or certain collections of users to have access to those. So maybe you don't want to give everybody in your organization write access to your core central data store. That would be pretty intuitive. But maybe there is sort of a secondary data store that you're okay opening up access to. And so you can do that.

So I guess what I'm saying is, today, the common path is that people connect proactively. And sort of one of the first steps of making Retool extremely powerful is that resource connection. And we, as a company, care a lot about sort of the quality of our connectors and how easy it is to connect to a resource, because it's such a powerful precursor to seeing how valuable Retool is. But we do have a solution, which again is more in line with how I built my my scrappy internal app, which is effectively, "Hey, you can sort of fully provision and fully manage a Postgres database directly through Retool." That's not live. For those listening, we're excited to get you all on board to that. But not quite live yet for everybody.

[00:16:27] JM: Not to open a can of worms, but how much mileage is there in that product strategy of spinning up infrastructure?

[00:16:35] SK: Well, it's not. I mean, it's still abstracted, right? I think that, for us, we can use best-in-class tools. We can basically have a new RDS cluster per customer who wants to use it. It's not that we're sort of doing it fully end-to-end behind the scenes, if that's what you're asking.

Because I think that that, while very interesting, is sort of a solved problem. But I do think, especially if you look at sort of earlier stage startups, I mean, this is sort of a canonical path of development. The first thing you do as a startup, someone who found a company several years back, is almost define your object model. And there's no reason why you couldn't do that from within Retool. And we could sort of give you a best-in-class experience there.

And that would be a very clear natural segue to then building internal apps on top of that, right? Said differently, I think part of the reason why internal tools are almost sort of like a lagging need in most companies, and oftentimes they build an approach when it's too late when the company already sort of has needed those tools for so long, and they canonically fall under sort of tech deb or product deb, is because you decouple, right? You define your object model. You never think to build an internal tool on top of it.

Retool bridges that world. It makes it really easy once you have an object model to just represent it to screen and sort of introspect it, engage with it, right? And that then allows sort of support processes to take hold. So I actually think, like, it's probably less right if you, I don't know, pick a logo from our website. If you're DoorDash, you're probably not going to be provisioning new databases from within Retool. And that's okay. But I think if you're if you're earlier in your journey, you may very well. And I think that could be interesting. But to be determined, right? We're still early in that exploration.

[00:18:15] JM: So when you're building a Retool app, there's a lot going on in the editor. And I'd love to know if there's any learnings you have from making that editor or that code builder performant.

[00:18:34] SK: Yeah, it's a really good question. I mean, to be honest, React has been very good to us. And today, I worry more about sort of the performance of how we evaluate or how we resolve our dependency graph, or where we have very large applications and you sort of see this sort of everything's reliant on one query, and that query happens to be huge. Like, I think there is basically a hierarchy of sorts of where we can go to extract performance. And as far as evaluating JavaScript, there's certainly intelligence that we apply today. There are certain opportunities to basically have effective guesstimation of what JavaScript is doing and decide whether we want to evaluate that in a sandbox, or they could just evaluate that in line, which is

obviously a lot cheaper. We have to share a lot less state and share a lot of scope in order to do that.

There's also just basic intuitive heuristical optimizations, like don't try and decode or encode everything and see what you can sort of infer without doing additional parsing logic. So I would say that, again, at a high level, we care a lot about sandboxing. Sandboxing is very important mostly from sort of like running untrusted code in sort of a secure environment. So that's an important precursor to us evaluating any templates and any code that's written. But other than that, I mean, I think we're at a stage right now where we are sort of working down the list of sort of most critical sort of performance problems that we're having. And candidly, a lot of those are in the application space, right? How we debounce queries? Or how we think about sharing state with the sandbox. And less about sort of a low-level evaluation, if that makes sense.

[00:20:19] JM: And are there any other areas where performance is has been an acute issue or has created interesting problems to solve?

[00:20:31] SK: Yeah. I mean, performance is an incredibly – I think you are hitting the right note here. Like, it's sort of an evergreen topic for us. And especially because as more and more folks use Retool, we see candidly applications that we could have never sort of envisioned. And again, unfortunately, no liberty to share the internals of those. But needless to say, we get some incredibly complex apps that are built that almost sort of stretch the envelope of what retool was conceived of initially for.

And so whether that's, again, separation of state and taking large, effectively, monolithic application and breaking them into multi-page apps so that we can intelligently load, right? Which is something we do today, right? If you're saying like a tabbed view and you're on tab one, we're not going and making requests on tab two in a blocking capacity, because we don't want to slow you down. And you can imagine that same paradigm applying to multi-page apps where we load intelligently and load lazily. But there's certainly like that dimension of it.

We also see – What's interesting that we see today is like a lot of different hardware is ultimately used to run retail. And as we've talked about, a lot of the compute happens on the client. So if you're on M1 MacBook, life is good. But if you're sort of on a – If you're long-tailed, you just

happen to be on a Chromebook, we really want to be thoughtful about how we are optimizing or rendering our compute so that you have sort of a high-quality experience.

And so in the future, I can easily – And again, I think this is why it's a really exciting space, whether it's sort of WebAssembly, or server-side rendering, or whatever other sort of channels that we want to explore, today, we're just getting started talking about those. That just sort of connotes how sort of excited we are for the journey ahead and how much more opportunity we have. But for the time being, again, I think our performance problems are very manageable to sort of our customer needs. And again, React has been very good to us.

[00:22:22] JM: Have you had to do any tweaking of React? Any kind of low-level work on some of the open source components?

[00:22:32] SK: Yeah. Not notably to be, again, totally transparent and honest. So I'd love to share some cool stories. But I don't actually have anything on that particular dimension.

[00:22:44] JM: Are you using kind of the latest, like, data manipulation tools? Like, I don't know, React Hooks, and kind of that stuff?

[00:22:54] SK: For sure. Yeah. I'd say, again, our application of React is very modern. But we're not – I was specifically commenting on sort of the low-level optimization and sort of going into sort of like the deep framework stack and sort of forking from there. We're not doing any of that.

[00:23:10] JM: Tell me about the management of the code base. How do you keep Retool itself manageable?

[00:23:15] SK: Yeah. Retool is, again, a really interesting company. I think we have about 35 engineers or so, which is pretty remarkable from where we are sort of as a company and as a product. I think about where I was in the past with sort of Lyft, for example, at 35, and sort of a different picture. And so, too, is sort of the code management.

I think for Retool, we're entirely in sort of mono repo land. We sort of manage those with Yarn workspaces. We look to share sort of – And that has advantages, right? We can share a lot of

code between our frontend, our backend, which is obviously simpler to do in a monorepo. And then the way we sort of create some separation of responsibility, so to speak, is just as you might expect with just the directory structure helping us separate different products and different product lines and different parts of our stack.

For us, I think it's really beneficial. And we've been able to sort of bring down the build times by sort of optimizations there. We recently sort of adopted Buildkite actually, which brought down our build times. But given sort of the size of the team, it's really nice to have everybody in all parts of the code base, I think, operationally that really helps in terms of on-call, that helps in terms of standardizing languages, right? You don't see sort of like language sprawl at Retool where there's sort of natural modes between services as you have to discover completely new technologies and languages. There's sort of a lot of interoperability both from the tech perspective, but also from the people side, which is really important to us. So yeah, hopefully that gives a little bit of insight into how we sort of manage and maintain the code base.

[00:24:45] JM: So does the code base have any mapping to how different teams are managed? Or is that kind of a disjoint question?

[00:24:55] SK: No. I think that was sort of the slight comment that I made nested in there about different directory structures. So you will see, for example, our sort of native mobile investment has its own sort of subdirectory. And again, we look to share as much code as possible so that we can be as productive as possible. But there are some clear departures in terms of what those products look like. So you see deviations as you traverse different directors. And and teams, for the most part – For the most part, right? Deal with sort of their directory and sort of the common shared code, but don't really descend into other teams. The file system basically creates separation that maps somewhat to our teams. That's, I guess, the TLDR of what I'm saying.

[00:25:34] JM: You mentioned mobile there. Can you tell me a little bit more about your approach to mobile?

[00:25:38] SK: Yeah. So I think we sort of have sort of two approaches, right? The first is we want to make apps that are built in Retool responsive. And there's actually a way to do that

today where you can effectively opt into responsive view. So that's wonderful from the perspective of if you're building an admin console and you want people to have access to it on the go, or on sort of unconventional devices like docked iPads, etc. You sort of have that solution out the gates.

We also think that mobile is one of those areas that there could be sort of mobile-first or mobile only teams out there that are looking to build internal tools for those efforts. So we have a sort of dedicated React native investment for that class of problems that is just jointly different, right?

What we want to avoid is people feeling like they need to rebuild the same application across different distribution mechanisms. So that's where that responsive, that first half plays in, where we have the dedicated responsive view. But then if you want sort of that that mobile-first and you want, I don't know, things like offline mode, you want native notifications, things like that, there's sort of that other pathway. But the intention is not to ever have you build the same app twice, right? That would be sort of, again, violating that constraint. And we suspect that if you were to build that app twice, you would go the responsive route.

[00:26:56] JM: And have you found any issues in making the form factor for – I think of Retool apps as potentially quite expansive. And they could take up a lot of real estate on a desktop. Are there any issues with making those kind of expansive apps amenable to mobile?

[00:27:23] SK: Yeah, it's a good question. And I guess you're asking specifically – Like, just to rephrase and make sure I understand, you're saying, "Hey, you mentioned earlier, if you've got these really big sort of like mega applications, how do you make those sort of viable for mobiles? Is that sort of the gist of the question?

[00:27:37] JM: Basically, yeah.

[00:27:38] SK: Yeah. Yeah, I mean, I think the honest answer is it's case by case. And again, this is where we try and stay somewhat unopinionated. Although we do want to help bridge that gap, that's where features like multi-page will really help. I think that the short answer is we want to put the decision in the hands of our customers, right? If our customers ultimately believe that their – I find it hard to envision this. But if their very large expansive app should be on mobile, it

should be viable on mobile. And maybe they choose to break it up into multiple applications today in order to make sort of the end user experience really, really delightful so that you're not scrolling through like 30 effective pages of content in order to get to the table that you were looking for. Or maybe they introduced sort of more elegant navigation to jump between the different parts of the application and they sort of use modules as a result.

Basically, what I'm saying is there's a path toward fragmentation if you really want to take one of those large apps and make it available for mobile. But yeah, I don't think that applying the responsive bit to some mega app is going to set your application for success. But I think that's exactly some degree the journey that the engineer building the application should be somewhat thoughtful about, right? And our position is that whoever is the builder for this application should have a general understanding of who the end users are. And if, indeed, mobile, it's sort of meant to be cross-platform and have multiple distribution strategies, the app might want to be structured in a way where you have modules and sort of sub-components and you break those out into sort of multiple apps that are stitched together through a unifying navbar equivalent. So that's sort of more, I guess, of a product solution to a technical challenge to some degree.

But I also think that's the right one. Because even if we perfectly optimized and everything was incredibly responsive and fast, you still are dealing with like limited real estate, right? You're visually constrained to a four-inch screen as opposed to a 15-inch one. And so there's only so much cognitive load you can push into that smaller form factor. That's what our builders need to be somewhat thoughtful about.

But like just to give you the counterpoint. Something like a form, right? If you're building a form within Retool, like, why would you need to spend any time having that – Spending any time making that work really well in mobile and making that delightful? And that's where sort of that responsive engagement makes a big difference.

[00:29:50] JM: Harkening back to the Lyft conversation from earlier, I think the mobile experience on a Lyft app is so informationally demanding. There's such a high flow of information between the client the server. And I can imagine internal applications where that would potentially be something that users would want. Like, I don't know, maybe a warehouse application, something like that. But I imagine most apps are not that data-intensive. Do you

have customers who have incredibly data-intensive apps? Like maybe even on par with something like Lyft?

[00:30:30] SK: Yeah. I mean, we actually do have some data warehouses and inventory applications that are being built. I think with Lyft, you sort of have – Locations is a really good example of just a massive stream of information that's consistently flowing. That might be sort of one discrete difference where we don't see sort of data streaming, to the best of my knowledge, at volume. But we do have incredibly high payloads. And we have folks pulling in information from data warehouse and sort of data warehouse scale. We also have the warehousing case, like inventory management for a physical warehouse. So I think you'd actually be somewhat surprised maybe and just if you looked at like average payload. Unfortunately, I didn't think to pull that data before I came on. So I don't have that sort of broken down by different application or customer segments and sizes. But it's an interesting thing to look at. I may actually do that after our call today.

[00:31:22] JM: Yeah. Can you use location services with Retool apps? Is that an application?

[00:31:29] SK: That's a good question. I don't know off hand is the honest answer. I'll be honest when I don't know things. I think, like, objectively, there's no reason why we wouldn't be able to. I mean, I think the challenge with location data is more about how that gets stored and managed. And again, that's sort of on the other side of the interface of the Retool boundary today insofar as we really just do what you tell us to do, right? If you want us to send something over any arbitrary wire, we're happy to do that. And then how you go and manage that, whether you drop it into Redis or put it in in disk, which seems like a mistake, and how you sort of manage sort of eviction and everything else. Sort of up to you, right? That's sort of out of the Retool contract, so to speak.

And so I think can we have the browser enable locations and emit locations and send it back? I mean, certainly. Why not? Whether we do today, I actually don't offhand, again. But I think that the complexity of streamed information is, again, outside of where Retool operates today. It's more in sort of the customer's backend, so to speak.

[00:32:29] JM: So you mentioned data warehousing applications. I guess are you saying that people are building a lot of like frontends for data warehouses, like a Snowflake application or a Spark application?

[00:32:46] SK: Yeah. I mean, we're seeing, first of all, a good amount. Again, this is – What's most impressed me about Retool in the relatively short time that I've been here is just, I guess, the cardinality of types of apps that are being built, right? And you see BI tools being built. You see Retool being used as sort of an incident manage visualization and sort of console of sorts, where it's backed by Snowflake, and they're running sort of models on top of Snowflake and displaying those in Retool.

Again, it sort of blows my mind sometimes the sort of types of applications that people build. But this is the elegance of the application today, which is that it's fundamentally stateless, right? Other than application state and what we store in terms of dependency graph. But in terms of customer data, we're not touching that. And I think that was covered in sort of the last podcast as well. And that just flows over the wire in a stateless way. And so, yeah, people build all sorts of stuff including on top of data warehouses. Bi tools, incident consoles, being just two sort of examples that lead to mind.

[00:33:44] JM: So there's this feature of Retool where you can basically have JavaScript running in most of the places in the app. You can just have arbitrary JavaScript. Tell me about the build path for getting that JavaScript parsed, and compiled, and running and just how you do that in a streamlined way.

[00:34:07] SK: Yeah. This is a little bit of our secret sauce. I'll keep some of the details to myself here. But I think that what I'll say – And this goes back to sort of, I guess, I didn't connect the dots as explicitly earlier. But for us, all templates and all templatized JavaScript is run in a sort of on-browser sandbox. And again, that's our mechanism of sort of running it securely. And so we sort of instantiate a fully evaluative environment. We actually share global state so a lot of the state from the app is mapped over to that sandbox so that you can traverse our graph and reference the other parts.

Again, what's cool about Retool is that, in that JavaScript, you can reference. Component one can reference component two, or can reference the sub-results of a query and really traverse the entire object model that's being generated from that query. So we share all of that state and then effectively hydrate run it within that sandbox. That's probably all I can say about it, unfortunately.

[00:35:08] JM: Do you use any GRPC handing data back and forth?

[00:35:14] SK: So we do support GRPC in terms of customers, but not within sort of the Retool app context. So again, if you have a GRPC endpoint, that is how you're serving data that you want to load into Retool. Full support there. But in terms of within the confines of Retool, we don't.

[00:35:32] JM: Gotcha. So you don't have any – I guess there wouldn't be any need for that just within the core application architecture itself.

[00:35:40] SK: Right. Right.

[00:35:41] JM: Is there a kind of a limit to a size of a Retool application? Are there places where it hits a threshold and gets unwieldy?

[00:35:53] SK: I mean, I think the short answer is no. The practical answer is somewhat, right? I mean, I think we've seen apps with upwards of 600 components just to give you sort of a general scale. We've seen apps with upwards of 20 queries, right?

I think the complexity is devil is in the details, right? If all those queries are select stars over tables with bad indices, which is, again, things that we don't necessarily control, you're sort of compounding performance implications to the point of the app not being super useful.

So I think this is, again, one of the really interesting things about building in the context of Retool is that there's a lot that we can do on our side to make our house, so to speak, as performant as possible. But at the end of the day, the end user runtime load time engagement is somewhat dependent on what's happening on the customer side. And actually, a poorly constructed Retool

– For example, you can imagine a world where every keystroke, you rerun some query. Is it conceivable to do that in Retool? Yeah. I mean, somebody could cue up the event handler and launch a massive sort of sequence of queries per keystroke. That would make the application grind to a halt. And it doesn't even have to be all that complex.

But what I would say is within the sort of range of reasonable use, we see some really beefy applications. We see ones where you've got dynamically nested tables, right? So you've got like a list view with a bunch of tables, each of which are custom written to support sort of embedded components that today we don't necessarily support out of the box. And those apps run. And they run relatively performantly.

So I think that there isn't sort of like that asymptote so long as it's within sort of the reasonable engineering principles. And again, I think this is what makes Retool really compelling and very powerful as an engineering tool, which is much like any language. Like you can mess it up, right? There aren't guardrails protecting you from yourself. And we ultimately love empowering our builders with sort of the full suite of tools in terms of language primitives that they can use.

The downside of that is there is the possibility that it's not a bad actor. It's just someone who might not have the same sort of fundamentals. And they could slow down the end user experience on their side. It has nothing to do with Retool per se. But it cuts across both, right? And that's the interesting part about performance. We want to do the best job we can. And we're always pushing on sort of how our world operates. And then you know we may even want – And the we talk about this, right? We want to give people observability tools. We want to run basic timings, right? And we want to expose all that to the builder and the end user to be able to say, "You know, this query is really long. Like, have you sort of looked at the underlying data model? Or, hey why don't you set up your own timers for different parts of the stack and then we will, through open telemetry, let you land those wherever you want to build sort of observability dashboards for Retool apps?" So all that is very top of mind. But again, I think that the short-abridged answer to your question is we see some really, really big apps, and they don't have problems. And that's exciting.

[00:38:53] JM: Tell me about the engineering management and, I guess, how the priorities are set and how decoupled it is. Like, if I think about something like Lyft, Lyft is kind of a pretty

effectively decoupled organization. You got pricing. You got mapping. You got routing. You got support, that kind of stuff. It's all pretty well-partitioned. Retool, again, I have a harder time envisioning what the kind of responsibilities, division of responsibilities is.

[00:39:33] SK: Yeah. I mean, for starters, Lyft lived quite well good. Great job researching and props. Yeah, I think for Retool, we have some obviously separation of responsibility. You can imagine that we have one part of our team that is really thoughtful about what is onboarding look like, right? So we believe that Retool is like learning a new language on some dimension. And regardless of whether you're learning it because you are sort of the zero-index person to discover it and want to learn more about it, or whether you're inheriting an application and you need to work in that context, right? That's no different than being joining a company and working on a Go service. If you've never been in Go, you've got to sort of learn on the fly.

So there's one part of our team that is very much tuned to that new user experience and sort of education and taking an app and breaking down its components and basically building dynamic walkthroughs and tutorials. Pretty cool stuff. We have one part of our team that is really focused on the different aspect. And I guess, that more accuracy is an org, with multiple teams about the different parts of building. So I think if you look at building within Retool, there's one part which is just sort of the UI. That's the layout engine, the component library, right? So that's one parcel.

We've got another, which is working with code, right? Everything that we've talked about, but take it even further. How do you debug? How do you introspect a model? How do you get auto complete working well, right? And so there's sort of one part that.

There's a whole other part of the team that is what we call our development tool chain. I actually kind of love this part, right? If you think about coding natively in your language of choice, you have all these surrounding systems that help you as an engineer be more productive. You have CI tooling, CD tooling. You have your IDE, right? You have sort of source control versioning, code review, peer reviews, right? You have source searching, source graph, for example. All of these products are really meant to make you more effective.

And if you think about it, the Retool abstraction prevents you from getting a lot of those. So a lot of those need to be sort of brought over from first principles into Retool. How do you test the

Retool app, right? And so we're building that. How do you version a Retool app to prevent just incidental changes from going out and source control it, right, so that you can roll back? You can move forward, right? You can have sort of different versions that you can pin to safely.

If you're – Again, pick any one of our large customers, and you're building a support tool that applies to a multi-thousand-person support org, you don't want to ship a bug every time you accidentally make a mistake. So there's real sort of complexity in what that development environment is. So we have a whole team dedicated for that. We have a team dedicated to all of our resources, resource support, resource quality, resource connectivity.

So I actually think, like, on the one hand, I completely hear you, right? You go into Retool and you just see this big canvas and all these components and all these features, and it's like it looks very tightly coupled. But I actually think you can split across the different dimensions of how somebody interacts. And to some degree, I think it's correlated with user segment or user expertise. Whereas maybe a more a more basic Retool user might not get exposure to all those teams that we talked about. I don't know if that sort of helps.

[00:42:33] JM: No, no, no. That makes a lot of sense. Given that you've laid that scaffolding, can you give a sense of the product direction of the company and what you're focused on today?

[00:42:42] SK: Sure, yeah. I would say it's sort of two dimensions. I mean, one is we're exceptionally proud of where Retool is today. We absolutely think we've solved a very clear user need. I think the sort of market is speaking loud and clear in that dimension. But we also think that there's a lot of improvement, a lot of opportunity for improvement.

I mean, I use the app, and I just see, right? The other day, I was playing with it, and model introspection just wasn't working. And so I felt almost like I was blind in terms of what do I de-reference? What is sort of my nested object called? How do I sort of extract that and pull it in? So there's sort of the long tail of improvements there.

I also think that every one of our products today has the opportunity for elevation. So on the one hand, we clearly know that there's collaboration happening in Retool as evidenced by our

investment in protected applications, and versioning, and branching, which gives you sort of, again, control over incidental changes or other people mucking with your app.

On the other hand, we don't have a very good story around multiplayer. And that's a very clear sort of opportunity and area of investment for us. So there's just a ton of opportunity to upscale the product as it is today.

I think we have probably the best and drag-and-drop engine that I've ever seen. But even, still, it has opportunity for improvement when you're playing with sort of nested components and moving things around between containers. And so I think that, again, there's just sort of like a water level improvement.

I also think that the other dimension is today – And we've sort of been talking about it a little bit, right? Retool is a great way to build incredibly powerful frontends that are massively accelerating and really allow orgs to become more agile and allow engineers to be more productive, or technical builders to be more productive, more generally. But I think we're really constrained in some dimension to the frontend. And like we were talking about with that database provisioning, we've got a couple more tricks up our sleeve where we basically expand our application surface area, so to speak. And we become – You can think about the other parts of the application stack, data layer, service layer, validation layer, everything that we sort of talked about, being part of what we exposed in Retool in sort of a low-code way. And so that's sort of the other dimension of our product roadmap.

[00:44:48] JM: As far as multiplayer, how important is that to have multiple people being able to edit an app at the same time?

[00:44:57] SK: Yeah, it's a really good question. I mean, what we're seeing today is that there are actually many cases where people are concurrently editing an application. Now, admittedly, I haven't done sort of the next level analysis to make sure that are they even touching the same components? Are they collaborating changes? Sort of what does that look like?

But I think if you draw a parallel to code, you have a very clear structure for how you rebase code against conflicting changes. And in general, any application, including an internal tool, has

multiple engineers. I think back to sort of the central tool at Lyft, which is an internal tool, I think had upwards of folks working on it. So the same way that you have sort of concurrency of edits in the repo.

At the end of the day, you sort of want to support that. And you could do that exclusively through branching. But again, that's another place we have opportunity, because our branching structure doesn't really help you do a great job in resolving conflicts, because it's a visual editor, and yet it's a textual dif.

But I do think that there is the potential for – And we see it today multiple people in an app, in an editor capacity at the same time. So displaying that, allowing you to sort of engage with that, I think is powerful.

To some dimension, I think like imagine if you were in a given repo or a folder within a repo and you say, "Hey, three other people are actively making edits right on either the exact file you're in or three adjacent files." If you didn't know that, you might want to, right? You might want to collaborate with them and be like, "Hey, I'm curious on what you're working on." So that we can sort of better design. And even serving that top level of you can see other players in the sandbox and other people in your editor I think can be really powerful. So I think we have early indication in the data that suggests that it is important. But we also sort of to some dimension won't know until we build it.

[00:46:42] JM: Just to wrap up, can you give me a sense of what would be the most ambitious place where Retool could go? Like if you imagine the platform in 10 years, what's a kind of crazy internal application that somebody could build?

[00:46:59] SK: I mean, if you want to go really crazy, you could argue that that internal constraint is what you should relax, right? And like, can we really get to sort of a pixel perfect, highly performant application that you can be proud of shipping to your entire user base and not just sort of for internal consumption?

So I think if you want to go 10 years out, I think that's sort of one thing. I think five years out, I would say that it really is, to some dimension, what I was saying about. If all the fundamentals

can be sort of dramatically elevated – And firstly, I don't think we need to be that crazy, because I think even just the product that we have today is incredible, truly. And I think its opportunity is our largest.

But if we could expand to the overall application stack and start to see these really compounding effects where people are building full stack applications within Retool, I think that would be wild, right? I sort of gave my example of taking some shortcuts. Again, I didn't have any sort of validation, rate limiting, access control, anything like that, queries, which is not the best way to build an app at scale. But if I could have built – If the moment you start to realize that, "Hey, I could actually build a full stack app in Retool in 45 minutes," which is sort of what I did. That's sort of a mind-blowing sort of moment.

And today, we definitely do that for frontend development. But I think there are other parts of the stack where you still have to bring your own, so to speak. But we could apply Retool principles and primitives to there to make schema, migrations and updates really easy. We can make sort of the repetitious parts of server-side development, and com platform, or payroll on top of Stripe that much simpler, and really allow you to sort of have that full stack functionality.

[00:48:36] JM: Well, Snir, thank you so much for coming on the show. It's been a pleasure talking to you.

[00:48:40] SK: Yeah, likewise. Thank you, and all the best.

[END]