EPISODE 1052

[INTRODUCTION]

[00:00:00] JM: Web development has historically had more work being done on the server than on the client. The observability tooling has reflected this emphasis on the backend. Lightweight clients, maybe you don't need that much monitoring, but the heavyweight backend, you need all kinds of monitoring and observability. There have been monitoring tools for log management and backend metrics existing for decades, and these tools have helped developers debug their server infrastructure.

Today, web frontends have much more work to do than they did in the past. There are detailed components in frameworks such as React and Angular and they might respond quickly without waiting for a network request with their mutations being processed entirely in the browser, and this results in better user experiences but more work is being done on the client-side away from the backend observability tools.

Matt Arbesfeld is a cofounder of LockRocket, a tool that records and plays back browser session and allows engineers to look at those sessions to understand what kinds of issues are occurring in the user's browser. Matt joins the show to talk about the field of frontend monitoring and the engineering behind his company LogRocket.

FindCollabs is a hackathon company that I have been building, and it is now free to create hackathons for nonprofits and schools. If you have a hackathon that you want to create for a nonprofit or a school, that's free to do. If you want to create a hackathon for your company or sponsor a hackathon, that's also an option. You can check it out at findcollabs.com.

[SPONSOR MESSAGE]

[00:01:44] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its

benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It's a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW]

[00:03:25] JM: Matt Arbesfeld, welcome to the show.

[00:03:27] MA: Good to be here.

[00:03:28] JM: Backend monitoring tools have existed for longer than frontend monitoring tools. The field of frontend monitoring is less mature. On the backend, you have lots of logging companies. You have metrics tools like Nagios or Prometheus. On the frontend, you have crash reporting tools as well as some basic log management. How does the domain of frontend monitoring compare to backend monitoring?

[00:03:57] MA: It's really much different. The way we see it, really, this whole space of frontend monitoring became necessary maybe 5, 6 years ago when really React, Angular, all these new single-paged application frameworks were coming into fashion and folks should have expected

more interactive web applications where when you click a button, it doesn't make around trip request to the server. You immediately see something happen on the frontend.
That shift has really necessitated a different way of monitoring client-side performance. Where are struggling on the frontend is just a much different domain than the server where it's more about tracing and infrastructure and server load. Where on the frontend, it's more things like race conditions or memory issues or even just user experience issues are more the prominent as well as just the range of browsers, devices, network conditions that you see on the client side. Yeah, much different between backend and frontend.

[00:04:56] JM: What you're saying there, that thesis is similar to the thesis behind crash reporting tools. I mean, the idea behind crash reporting tools is a user has a crash on their device and some client-side stack traces created and shipped up to the cloud. Then if I'm the developer working on this app that just happened to crash on the user's device, I have the crash report and I can debug from there. But I think what you're observing is that it's gone from white and black levels of, "This is a bug. This caused a crash," to things that are more like performance issues, race conditions, jitteriness. Things that are less starkly delineated as bugs.

[00:05:45] MA: Exactly. Even a crash, you have different types of crashes. You have one that breaks the entire experience. Think of your mobile app that just crashes completely. Then more sort of the silent crashes where maybe the browser can retry and it doesn't even appear visible to the user. I think even now with the way single-paged apps are architected, that kind of notion of crash or not crash. There's a lot more gray area, and that's where client-side monitoring is really much more different than backend monitoring.

[00:06:15] JM: Can you say more about why that standardization around frontend frameworks like React, and Angular, and Vue. Why have these single-paged application frameworks made it more necessary to have better frontend monitoring?

[00:06:34] MA: Yeah. I think the way I kind of see it is where is more and more code being written. In a PHP stack, for example, 95% of the code is backend code and it's doing all the processing. Now, it's maybe closer to 50-50. It depends on the application. You have a lot more frontend developers. Often, it could be more junior developers, not always. More and more code

is being written on the frontend, and that happens if people are writing React and Angular. But just the more code that's there, the more surface area of what can go wrong.

I think those also, again, not based on the frameworks, but more user experience expectations where people are so used to their Facebook and Snapchat and really smooth interactive applications that a lot of the more advanced techniques and frontend engineering are to support those types of user experiences that maybe 5 or 6 years ago was just not the expectation. I think increasing user demands on the experience of the apps as well as just more code being written is what really makes it different than 5 years ago.

[00:07:40] JM: You spent some time working on the Chrome team when you were an intern at Google. Does your experience having worked on Chrome, does that play a role in how you think about frontend monitoring?

[00:07:52] MA: Definitely. I was at Google at Chrome. I was also at Meteor, which was a JavaScript frameworks company and worked on the frontend framework there. The feature I was working at Chrome was for canvas, and really there I was working with developers on the cutting edge of building JavaScript applications, apps like Google Maps and gaming where so much code was going to the frontend and just saw this whole emerging world of what people were trying to do when it came to web applications. I'd say, really, Meteor was where I saw the most sort of need for client-side monitoring given how people are building apps and how developers really had no visibility into what was going on on the frontend.

[00:08:35] JM: A slight deviation on the Meteor front. Meteor was this full stack JavaScript framework and a company built around that for hosting it. I used Meteor for a while, and it was an amazing experience. I felt like they got caught at an interesting time where the market was shifting and actually React came out and kind of disrupted the thesis of the company, because I think the thesis was you want this unified JavaScript development experience. Then with React, you had a really good Vue layer that everybody wanted to use. Then it kind of disrupted the idea of Meteor, because no longer did people necessarily want the full stack. They wanted to use React on the frontend than do something else on the backend. Do you have any reflections on the way that frontend developed from your time at Meteor?

[00:09:35] MA: Yeah. I think that's the right analysis where a lot of Meteor was about this kind of coupling of the frontend and the backend and write code that works everywhere. I think it was an ambitious project, but it ended up being a team that was just focused on the Vue layer like React and a team that was just focused on, say, data fetching, like the GraphQL team and node on the backend, that those technologies should have stacked together in a way that was maybe more effective than this monolithic framework.

I think what the team realized overtime was where really the challenges is how do you keep data consistent across all these layers, and that's at least why I believed they've moved more towards GraphQL, and now Apollo, where really data is at the center of it instead of saying, "We want to write code that works everywhere."

[00:10:28] JM: Okay. If we skip to the present, we talk about common problems on the frontend that make it into production, problems that the user is going to be experiencing in their browser. What are the issues that a developer might not identify in development that would make it out into production?

[00:10:50] MA: Yeah. Working at LogRocket, you see every millions of types of issues that might happen to a customer, and so few of them you can actually be captured when it comes to testing or integration testing. It really runs the gamut from a user is just confused. Usability type issue, where maybe you thought something was clear, but the messaging was not clear. Seeing issues where the user types in something and the validation returns an incorrect validation, and that's not clear to the user.

You have issues where there are certain paths that users take that you weren't expecting and weren't tested against, but something like an exception happens or state gets messed up. Depending on the app, you might be using local storage or session storage or cookies. When you do an update, you're not respecting kind of the old version of the state in the browser. It could go on and on. Memory leaks, crashes, and that's just sort of explicit issues. Then there's the whole world of performance problems that adds another layer of complexity on everything. It really can get quite extensive.

[00:11:53] JM: LogRocket is the company that you founded and it's a frontend monitoring tool. It records user sessions. Let's say I hit a page with LogRocket like Instacart, for example. I'm shopping on Instacart and LogRocket is also doing something in my browser. What's going on there? How is it recording my user session?

[00:12:18] MA: Yeah, good question. So what we do is we use an API called mutation observer in the browser, which basically lets us know whenever there's DOM that changes on the page, we're informed of those changes and we capture essentially the diff of all the changes that are happening in the Vue. Let's say you click a button and a modal pops up. We're recording that a modal just appeared, serializing that and sending it to a server. Then let's say you have an issue. A developer at Instacart goes and views that session, they're able to essentially replay exactly what you're seeing in the browser. Go down to the mouse click and that modal that appeared in the error message that you saw.

[00:12:58] JM: Does that API for mutation, does that also handle mouse movement tracking?

[00:13:05] MA: That's separate. There's another API in the browser that essentially will tell you anytime the mouse moves. We use that API and then essentially sort of compute the path of the mouse took without sending millions of events. We compute sort of the best path that we think the mouse took. Also, events like hover, focus events, so that we can make it appear when you're replaying the session where that mouse was in real-time.

[00:13:31] JM: I have my own opinions on this. The obvious question that people think about with these session recording tools is whether or not it's a privacy problem or under what conditions it's a privacy problem, because the reality is that like if I go on Facebook, yes, Facebook has all of the information about my sessions and I'm sure they have tools that they could use to replay those sessions. These kinds of tools are a reality, and if we plug our ears and close our eyes, that's not going to prevent the fact that these tools exist all throughout the Internet. But is it a privacy problem, or how do you interpret that fact that we now have tools, widespread tools, that observe the entirety of user's browser session?

[00:14:20] MA: Yeah. Definitely, this comes up a lot, and we think a lot about this here at LogRocket. The way we see it is we want to do everything in our power so that you're capturing

just the information you need as a developer to troubleshoot issues and not information that's creepy or unnecessary. There's definitely an effect when you're watching someone that I can appear creepy. The big things we do are essentially when you're debugging an issue, you probably don't need to know every single text the user is seeing, everything they've typed into every form. Allowing you to basically sanitize all the information by defaults and just enable certain fields or data that you want captured so that you're just explicitly choosing what data you need to troubleshoot issues.

Giving really fine game controls around what's captured, what you actually need to know as a developer. Then the other part is your big problem is when let's say your healthcare company and your customers believe I'm just interacting with this healthcare company, but yet their data is going off to a third party, that being LogRocket. We've put a lot of effort into our self-hosted version where if you're that healthcare company, you can store all that data on your own servers where you're already capturing all that data so there's not another third-party or a source where that data may go. It definitely has privacy implications, but we kind of see our places. Let's do as much as we can to make it as protective as a user as possible while still providing the value to developers.

[00:15:45] JM: All right. Well, maybe we can revisit that later, but I want to have more about what this actually does to provide value to developers. If I am using Instacart, I'm shopping for fruit. I add some grapes to my shopping cart, and then the grapes disappear from my shopping cart. I email Instacart support. I ask them why did the grapes disappear from my cart. What's going on here? How can they troubleshoot an issue like that?

[00:16:19] MA: Yeah, that's a good question. Typically what will happen is Instacart will have some sort of support tool like a Zendesk intercom and LogRocket will appear in that chat when you go chat in. Then when you view the session, the developer will – They'll likely see some sequence of events that's unusual. So maybe it's an order in which you clicked buttons. Maybe since we capture the API requests as well, maybe a certain API request was very slow or didn't complete. They're able to view that information and see what was different than what you'd normally expect to happen.

Usually, they're using the sequence of clicks or mouse movements from the user, the API request, or if you're using a framework like Redux, we also capture the Redux actions. Those are the main data sources a developer would use to figure out the root cause and then reproduce it locally.

[SPONSOR MESSAGE]

[00:17:15] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:19:04] JM: If the user is navigating Instacart and that session is getting recorded, the mutations to the page are getting recorded, the mouse movements are getting recorded. Can

help me understand better how that data interchange between the client and the server is working? How are the recorded sessions – What format are they being packaged as and sent over the wire to you?

[00:19:28] MA: Yes, good question. We use a format called protobuf, which is a binary data format invented by Google, and what that does is where if you think of a JSON data, a lot of the storage there is in the actual keys. That can really add to the amount of data you're sending. What protobuf allows you to do is instead of having to pass the keys with every single data that you send, it just is sending values and then a sort of indicator of what data type it is.

Instead of when you send a mouse movement, instead of sending X, Y, and the position of the X and Y coordinates, we can just send the coordinates and it knows how the data should be represented. Protobuf is the main format we use, and it helps compress the data up to 90% when we're sending it to the server.

[00:20:15] JM: How bandwidth-intensive is it, like even after the compression? Is it severe enough to ever cause performance degradation?

[00:20:22] MA: It depends on the application and how much data is being captured. Most apps is around 2 to 10 kB of data per minute. It can be more than that if you're capturing, say, large Redux stores. We also dynamically – If we notice it's a 4G connection or the Internet is really slow, we drop certain pieces of data so that we're not hogging the network bandwidth.

[00:20:45] JM: Okay. There are some issues that can be identified just by looking at the browser behavior and looking at the mutation changes that you look at, but fronted issues can also result from networking problems. Do you record the networking behaviors as well?

[00:21:03] MA: We do. Yup. We basically proxy the fetch request and XHR requests to capture those data payloads. We also have built such a way to determine if assets are failing to load, which actually a shift fairly difficult problem. Those are a really good source of data, especially in the modern single page app where API failures are a big cause of issues.

[00:21:23] JM: You said you proxy the XHR request. Can you explain what that is in more detail?

[00:21:28] MA: Yeah. To install LogRocket, developer installs in npm package, and when you import that npm package, the awesome and the evil thing about JavaScript is that you can basically overwrite any function that the browser uses. What we do is we overwrite this API called XML HTTP request so that when the application calls into that API, it's actually calling our version of the API and we take that data and then send it to our server as well as allowing that data to pass through. JavaScript really allows us to do anything we want and as long as we're imported before the application runs.

[00:22:07] JM: Okay. Great. Well, I think at this point we have an overview of how you've built a frontend monitoring tool at least on the user side and on the side of I install this thing and it sits in the user's browser. I want to know more about your backend and what you're doing to take in all of these user sessions and how you're storing them and what your infrastructure looks like. Give me an overview of your backend.

[00:22:36] MA: Yeah, definitely. Like I said before, a big part of our solution is that it can be deployed self-hosted in, really, any environment. From day one, really, we've been building on top of Kubernetes, which makes it very portable applications. We run in Google Cloud, but we have customers who run in Azure, AWS, even bare-metal Kubernetes infrastructure. Kubernetes is really the core of our of our stack, and then we use Postgres for most of the data storage.

[00:23:06] JM: What was the reasoning behind starting with Google Cloud as supposed AWS?

[00:23:11] MA: That's a good question. I think just one day I woke up and decided Google Cloud seems cool. Then when we first started, they were very generous with credits for startups and we found the UI to be more intuitive. I wish there was more rhyme or reason behind it, but it was just a place that seemed best for us.

[00:23:29] JM: Well, it seems like it's almost becoming like an operating system choice where it's like if you go with – I mean, I don't want to put words in your mouth, but if you go with Google Cloud, it's more of a clean room of infrastructure tools, whereas if you go with AWS, you get all

of these bells and whistles and these amazing integrated tools that work really well together. But the interface has really got a lot of stuff going on. If you kind of know what you want to build yourself and you don't really need like Lambda functions interacting with each other and the full stack serverless buffet of things. Google Cloud seems like a better, cleaner option.

[00:24:10] MA: For sure. There's definitely more services in AWS, but we don't really need satellite scanning for what we're you doing now.

[00:24:18] JM: Yet.

[00:24:19] MA: There's enough in Google, and they keep building more and more every year, but there's definitely some amount of bloat and it's overwhelming when you open the AWS options and there're 200 different tools and services available.

[00:24:33] JM: As far as building on top of Kubernetes with the mentality of we want to make this thing portable. We want to make this thing deployable to enterprises that want to self-host it. Is there anything tricky about that? When you were creating the self-hosted version and getting that to be stood up on Kubernetes, any unanticipated complexities?

[00:25:00] MA: Definitely. I think it the whole dream of Kubernetes is that you can do something like this and you just with one click move it between clouds. But I think we're still probably 4 to 5 years away from that vision completely.

Some of the things that are harder, auto scaling. Yeah how do you make sure things scale correctly across different clouds? How do you deal with data ingresses? How do you get data into the system in a consistent way across clouds? How do you set up underlying infrastructure if it's not within Kubernetes? How do you manage a database if it's external to Kubernetes, or do you manage that internally to Kubernetes?

We still run into a lot of those types of problems today; certificates, but definitely it's a lot easier than the old days where you have to create a VM, configure it and all this stuff that Kubernetes just takes care of for you.

[00:25:49] JM: In building that yourself, do you have some way to simulate what the enterprise infrastructure would look like or do you just have people that you – Like customers, potential customers, that you have try to deploy it with their own Kubernetes and then you kind of place a support role and just help them triage through those issues?

[00:26:13] MA: Yeah, more the latter, and I'd say it, fortunately, most Kubernetes environments are fairly similar in terms of what's available. What the APIs are? Is it a standard, so it's not really Kubernetes compliant unless it follows those standards.

I'd say if the customer is running Kubernetes, it's a fairly standard deployment path. Then for customer not already running Kubernetes, we have terraform scripts that will spin up sort of a terraform environment that we're comfortable with.

[00:26:41] JM: You mentioned Postgres is your main database. What's behind the Postgres choice?

[00:26:48] MA: Yeah, that also was just an early decision that probably still no rhyme or reason behind it besides it worked and was able to scale pretty well. Good amount of our data is relational in nature. Being able to do joins and transactions versus more of a NoSQL option, but definitely you get into more limitations as you scale Postgres and more complexities there.

[00:27:09] JM: Tell me more about the ingress and storage process. You get these user sessions in. They come in over protobuf and you've got to deserialize them. You got to do some things with them. You've got to figure out how you're going to store them. Tell me more about the ingress and storage process of all these user sessions.

[00:27:29] MA: Yup, exactly. Like you just described, you have to store the data. You have to, for different data types, transform them in different ways on the backend to be in a format that we can then replay.

A good example of this is let's say you're in Instacart and you view an image of the grapes, that image may change in the future. So that URL may change. We actually cache that asset on the

backend so that when you go replay that session, we have the same exact image of the grapes that the customer had when they were interacting originally.

[00:28:02] JM: Wow! The actual session data, is that being stored in Postgres? Are you throwing that in S3 or something?

[00:28:11] MA: That's actually stored in Postgres. Yeah.

[00:28:13] JM: Interesting. Do you just store it as raw deserialized, replayable log data?

[00:28:23] MA: Yup, exactly. The protobuf format is nice because it's fairly small data and Postgres now has fairly good ways of managing just binary data types. It's worked for us here. Definitely, in the future, we may have to explore other database options, but at least for where we are now, it's worked.

[00:28:43] JM: Like under what circumstances do you think Postgres would break down?

[00:28:46] MA: Yeah, there's definitely issues with scaling and managing larger Postgres instances. I think it's more a function of how much data you're ingesting in there. But we manage hundreds of terabytes of data a month and it's worked well for us.

[00:29:00] JM: If the user or let's say the developer who has gotten this crash report or this bug report, they login to the LogRocket developer terminal and they can load up the problematic session and they can replay it, what is the process of loading that session from the protobuf data that's stored in Postgres into the replay environment? How do you like hydrate that user session and turn it into this robust replaceable session capture?

[00:29:43] MA: Yeah, definitely. That's where so many edge cases when it comes to this, because we're essentially rebuilding a browser within the browser. What we do is when you're replaying, we actually have an iframe embedded in that developer console where we re-create the DOM elements as they appeared to that customer and synthesize a lot of the events that were going on.

If you were hovering over a button, we actually insert this kind of fake hover CSS class that so that it appears like someone is still hovering over that button. But really we use that iframe to recreate what the browser looked like at that time. Then as you play or as you scrub in that session, we apply the diffs on that Dom data to get it back into the state that the customer saw. That's where a lot of the complexity is, is how do you organically recreate what the user was seeing and also do it performantly and quickly so that you could go an hour into a session and it shows up in milliseconds for the developer.

[00:30:44] JM: Yeah. That's what's kind of crazy, is if I think about the full stack of what we're talking through here, like the user has gone to instacard.com and LogRocket has first grabbed schema of the page. Then as the user is going through their session, LogRocket is capturing the mutations to the page and shipping those diffs. Then at the end of the session, you've got basically the page and then the change set stored in your Postgres database. Then when the developer goes into the console or into the backend developer experience, you're creating this way of them being able to scrub through the entire session like a Netflix movie. That means you have to be able to re-create the entire page in an iframe and then be able to scroll through the mutations to that page in the iframe.

Can you just give me an example of why that is hard? Of why it's hard to recreate an entire session in an iframe?

[00:32:00] MA: Yeah. One element is there's so many different browser APIs that you have to account for that when in a normal session you don't have to consider those, but then when you're replaying, there's so many edge cases that you have to essentially rebuild. A good example is Shadow DOM, where to capture Shadow DOM requires you to actually add another mutation observer to every Shadow DOM elements. The when you rebuild it, make sure you respect the namescaping of the Shadow DOM CSS so that you don't have styles leak into the Shadow DOM elements.

Another good example, if you think about, say, a two-hour session, there might be tens of thousands of changes that have happened. But a developer, the problem may happen at the end of that two-hour session. How do you enable someone to go view the second hour of the session without it taking minutes and minutes for that session to load? The performance

implications behind making that data available as quickly as possible, that's a difficult technical problem.

[00:33:02] JM: A lot of what you're talking about, these problems seem like things that you would not be able to anticipate until some developer that was troubleshooting an issue, they were using your troubleshooting tool and they're like, "This doesn't look right."

[00:33:19] MA: Yeah.

[00:33:21] JM: Can you tell me about like what's that feedback process? I mean, I can imagine kind of an irate customer who is like, "Your debugging tool has bugs in it. I'd like to file a ticket."

[00:33:32] MA: Yeah. I mean, even worse than that. The first customer we on-boarded, I was at their office. They deployed our SDK to production and immediately they saw all their analytics dropped to zero, the entire set of crash. Years and years of beta testing and just onboarding customers until, overtime, it's like a whack-a-mole. You have enough customers report issues that you can go and figure them out. But I remember when we were first starting, like first six, nine months, we would just ask customers, "Can we view your sessions?" We'd go in, inspect what looked off. Do a deep dive into it, and then we have a Chrome extension that actually lets us inject our script on to their site. We would go figure out what the edge case was that we didn't account. Write a test for it and then fix. But that process takes a long time and went through iterations of that to get it right.

[00:34:24] JM: This whole space is pretty interesting, the frontend monitoring space, because it is like one of these spaces that when you begin to look at it more closely, it's very deep. I mean, the same thing has been happening on the backend over and over and over again where you have every new generation of new platforming. Platform infrastructure leads to new logging companies and it leads to new monitoring companies, for whatever reason, and nobody ever throws out their old log management software as far as I know, or if they do, that migration process is very, very slow.

I'm wondering, from the business perspective, how do you anticipate the frontend monitoring space unfolding? I mean, there're really only a few players at this point. Do you think it is as deep as backend monitoring?

[00:35:17] MA: Yeah, absolutely, and I think it's deep in a different way where in backend, if you look at some of the great companies out there like Dynatrace and Datadog in AppDynamics, they do so much intelligent around where our API request going and how you're tracing it and how does our service dependencies mapped? How do we trace in 50 different languages? How do you log data and make it affordable?

On the frontend, I think it's different, where it's more around what are the problems that are really impacting customers and how do we surface those effectively. That's really where I see the space of the frontend monitoring going, is how can you better filter through all the noise on the frontend to show developers and product managers and designers what's really important to the customer experience.

[00:36:02] JM: We talked through a workflow of the customer actually saying what went wrong, like I'm going to send an email to Instacart that says, "Hey, the grapes didn't make it to my shopping cart." That's never going to happen. I'm never actually sending that email. I'm just going to like find a different fruit and add it to my shopping cart or add like the white grapes instead of the purple grapes.

You mentioned surfacing these issues, because a lot of these issues are just like little jittery things. It's like nobody is going to report this. How do you identify those issues without requiring a human to intervene?

[00:36:43] MA: Yeah. There's a few different ways that we're approaching this. One is how are users responding to that red grapes? If you try to add the red grapes, you probably not just going to try once. You may try 2, 3, 4, 5 times. How do you identify those patterns of user frustration and use that to better determine what's impacting customers. That's kind of one vector to look at the problem.

Another, we do have thousands and thousands of users who do report issues and the developer goes in and looks at them probably more for business apps or apps where you really engage and you want to get something done. We can look at those users who are actually reporting issues and use that to determine what is user frustration on the web actually look like. It's kind of one side of user frustration.

The other side is looking more at business metrics. If you're Instacart and you're tracking what percentage of users who add in items to a cart, then check out. What are the types of problems that are causing that conversion rate to be worse and use that to better determine what's actually impacting customers? Those are a few of the different techniques that we're looking at to better proactively surface issues that matter.

[00:37:54] JM: The second example you gave, like they were looking at factors that affect the conversion rate, like the time it takes to add red grapes to my cart or the number of times I have to try to add red grapes to my cart before they actually are added to my shopping cart. I mean, I'm just trying to understand. If I know some information about what does or does not lead to conversions, how is that going to change my workflow of sifting through – There are millions of people using Instacart. They use Instacart three times a week. That's 3 million sessions. How is Instacart going to use their knowledge of what leads to conversions to sift through all these sessions actually prioritize what to study in all these sessions?

[00:38:41] MA: Yeah. One way to look at it is let's say you have an error that's just nor is and not affecting the user at all. Users who have that problem versus users who don't, they're going to convert at the same exact rate, because it's not affecting their experience at all. But the user who tries to add the red grapes but fails and sees never, they're probably going to not convert as much. They may go Peapod or another food delivery service instead. Comparing the users who encounter the problem versus not and seeing how those folks differ. That helps our algorithm on the backend better show what issues are important to the customer.

[SPONSOR MESSAGE]

[00:39:28] 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]

[00:41:05] JM: You mentioned earlier that in some cases, you can take the Redux store and you can snapshot the information in the Redux store. If the site that the company, like Instacart in this case. If Instacart was using React and they're using Redux. If I recall correctly, Redux is like this in-memory thing that just sits in your browser and basically you can throw stuff in it. As the user navigates through all these different pages, you have this sense of shared state that sits in the Redux store. How does snapshotting the Redux store? What kinds of advantages is there to doing that and shipping that to the user session playback?

[00:41:45] MA: Actually, the first version of LogRocket, we were inspired by Redux to build it, because Redux is kind of this log of every single thing that's going on in the frontend where when the user clicks a button, that triggers a Redux action that you could then serialize and sent to a server. We saw that and said, "Actually, this is really useful information for troubleshooting. Basically, using all those actions that the user takes, that can help the developer narrow down to what exactly was the user sequence of events that caused this problem to occur and then even go – Let's say there's an error state. You can even take that same Redux state and bring that back locally to reproduce the state more quickly that way. Redux really provides this nice

serializable data set that you can use to figure out what exactly did the user do to get into a certain state of the application.

[00:42:38] JM: Was your initial product completely focused on React developers?

[00:42:42] MA: Yup. Yeah, first version was all just for Redux, and there was no video playback at all. It was just the series of Redux logs, and kind of of as two people do just working on their apartments. One day, someone decided, “Oh! What if we add this session playback piece?” We did that, and that's kind of how that whole piece started. Originally, just all focused on Redux and react to developers.

[00:43:08] JM: The other company I know of in this space is FullStory and they do something similar. Do you know how their engineering, their implementation of session capture, compares to what you're doing?

[00:43:21] MA: I'm not super familiar with their exact implementation, but I think most companies approach it in a similar way these days with using mutation observer to capture the series of DOM events, but I'm not exactly sure of their specific implementation.

[00:43:36] JM: Is that a newer API? That mutation observer?

[00:43:39] MA: Probably about 8-years-old. So, yeah, on the newer side, but not sort of cutting edge.

[00:43:44] JM: Got it. Got it. These frontend monitoring tools, they started to come out. Like you said, this was necessitated maybe five years ago, but it seems like the frontend monitoring tools, they are much newer, maybe three – When did you start LogRocket? Two or three years ago?

[00:44:00] MA: 2016. Yep. So about four years now.

[00:44:03] JM: Okay. Well, I guess that’s not that long, but you could of had this kind of company come a lot earlier, right?

[00:44:10] MA: Definitely. I mean, the original real inspiration was I was working on our frontend, the backbone frontend and users would report problems, send screenshots to my CEO, who then forward the emails to me and I have no idea how to figure out these issues. That was probably 2015 backbone. But I think now it's a combination of more – There is enough frontend apps to support a business like this, and the performance of browsers is good enough that doing this kind of session capture, especially the amount of data we collect, is performant enough. I think now is really the – Or 2016 was really a great time to start the company where this is a wave that we think where this continue to grow.

[00:44:51] JM: The consolidation around frontend frameworks, most prominently React, but there are people – Who if you know Angular really well, you just dig in more. You get a custom Angular. You have the Vue cadre of people who just love Vue for whatever reason, different design decisions. That kind of standardization around frontend frameworks, are you seeing the same kind of standardization in GraphQL? Is the frontend community really centralizing around GraphQL or do think it's still more, more of an optional kind of thing?

[00:45:30] MA: Yeah. It's definitely growing in popularity from what I can tell, and particularly Apollo and kind of the shift from Redux to Apollo and GraphQL, but I'd still say it's on the early side in terms of adaption. Whereas I think React really reduces the complexity in a lot of ways. I think Apollo and GraphQL can add complexity in other ways. There's more tradeoffs consider, whereas comparing React, say, jQuery. It's really hard to find downsides. I think there're more tradeoffs to be had when you're thinking about GraphQL versus REST.

[00:46:04] JM: Do you get any macro – I don't know how many customers you have, but I'm imagining enough customers to have a pretty big sample of what frontend stacks look like today. Do you have any macro observations about how frontend stacks are changing or I guess just JavaScript stacks are changing?

[00:46:29] MA: Yeah. We haven't done too many sort of macro projects, analyzing all of our customers that we do want to do those and put some of the, obviously, anonymized data on our blog at some point. I do look at the npm stats quite a bit in comparing React packages, Redux, GraphQL, and I feel like those show a good representation of how these stacks are getting

adapted and all that, but we have more data around sort of what issues customers are encountering about it than looking too much on the frontend stacks.

[00:46:59] JM: Okay. Well, just a few more questions on go-to-market and just kind of take a step back. What is the go-to-market like for a frontend monitoring tool? What strategies for building user adaption have been successful for you?

[00:47:17] MA: Yeah. Both my cofounder and I are developers and we've always found products from searching for problem, finding a solution and implementing it. That's really where we started from day one. Hopefully some of the listeners have seen the log market blog, where we've invested a lot to look to build a really great resource for frontend developers, and hopefully they use that, and when they do have a problem that we can help solve, that day they find us. That's been our first and foremost way of finding customers and building community.

Until coronavirus and COVID, we also went to a lot of different sort of conferences and meet ups and tradeshows where frontend developers were spending time. Obviously it's harder now with the current atmosphere and environment. Then a big thing for us is also just word-of-mouth and referrals and asking happy customers to tell their friends. Really, the blog, and we started that. Basically, the same day we started the company, that's been the biggest source for us.

[00:48:12] JM: How do you see yourself positioned relative to the crash reporting, the crash monitoring tools?

[00:48:21] MA: When you look at most of the crash, really, all the crash flowing tools. They started at monitoring like your Django application, your Ruby on Rails application and then moved into the frontend and kind of applied similar techniques to frontends as the backend stacks. They work great for capturing [inaudible 00:48:38] exceptions and merging and making those actionable.

We kind of see ourselves as what are all the problems that are not, say, crashes that are happening to customers and making those actionable and solvable. We were complimented quite a bit by products like Bugsnag and Sentry, and I think the best stack involves incorporating us with those other products.

[00:48:59] JM: Interesting. That workflow would be like some crash monitoring tool detects a problem and ships a notification to Pagerduty or whatever. Then that developer needs to figure out what went wrong, and then they dig into the session playback in something like LogRocket to fully triage the issue.

[00:49:26] MA: Exactly, or there's a backend problem, but it's not obvious what caused that API request to be made. Then you can view the session and see what happens. Then we also go the other way where if a user reports an issue and you detect something is wrong, you can then go and go from LogRocket to a tool like DataDog or Splunk to figured out what happened on the backend. We're trying to really connect all the tools as best we can.

[00:49:50] JM: Right. That opens up a lot of questions. If I'm troubleshooting something that occurred on the frontend, it is often going to require this piecing together of things that might've happened in the frontend and the networking layer and the backend and trying to get a unified perspective for this workflow is not easy. It's not as easy as like, "Oh! There's some magical distributed tracing tool that's going to piece together everything that occurred at each of these layers." It seems like the only way for a developer to really triage these kinds of issues is to look at them from a number of different angles and then poke and prod at it, right? Tell me more about like workflows of using different toolsets of diagnosing problems that might involve errors at multiple layers of the stack.

[00:50:45] MA: Yeah. We kind of see it as LogRocket. We can capture all the data that's going on the frontend, but backend visibility is a whole another beast and ballgame, and that's where we want to bring you to that data on the backend.

An example would be you say, "Hey, I try to buy grapes. It didn't work," and there's a specific API requests that timed out. We can then enable you to go from that request within LogRocket and see all the associated backend data within Datadog from that request. What was a trace that associate with it? Why was the database call that failed? But in that case, LogRocket is really the entry point into seeing where in all these backend data should I even start looking?

[00:51:25] JM: Got it. By the way, when I was preparing for this show, I watched some stuff on YouTube, and it seems like you do have a pretty good content strategy because you got a lot of content on your YouTube channel. I saw a lot of like pretty good blog posts on your blog that looked like they were sort of outsourced. It looks like you got a pretty good content strategy. But one thing that was kind of hilarious that I saw interfering with your YouTube SEO, there's apparently something called a LogRocket stove.

[00:51:55] MA: Oh yea! That's our favorite. Everyone should Google LogRocket Stove. They're somewhat dangerous. So be careful. But that's our favorite company pastime, is building LogRocket Stoves.

[00:52:07] JM: So you've actually done it?

[00:52:09] MA: No. I actually haven't, but we've always been planning a company outing to go camping and a LogRocket Stove competition. That is an unfortunate competition to our SEO.

[00:52:18] JM: Yeah. It looks like there's a full YouTube subculture of people who build – I guess it's literally like you carve out the inside of a log and then you turn it into a stove.

[00:52:30] MA: Yeah. It's pretty impressive. I guess you need to bring a drill out there and tools to accomplish that, but it's better than a campfire I hear.

[00:52:38] JM: Well, I don't know the advantages too much, but I was thinking about it a little bit, and like if it was raining, it will be pretty useful I think. They could start a fire in the rain. Anyway, this is not like camping engineering daily. That's for another episode.

What changes to the frontend world do you expect in the next couple years?

[00:52:58] MA: Yeah. I'm pretty excited about WebAssembly and the implications there. I could see that really enabling a whole new class of web apps and more gaming, more sort of apps that traditionally would have to be desktop moving to the browser. I think Figma, if anyone is familiar, is a good example of really an app that's been enabled by that shift.

The other area I'm interested to track is how mobile develops, and for years, everyone has been saying mobile web is the future and every mobile app would just be a web app, but seems like potentially it's shifting actually more to native in the past few years. I'm curious if that swings back and we start seeing more and more web apps on mobile the next few years.

[00:53:41] JM: The WebSssembly story with Figma. If I recall, Figma ships entirely as a C++ app and then it just runs in the browser, which is kind of crazy workflow.

[00:53:55] MA: Oh yeah, and that's the really interesting thing, is that you're not necessarily going to code in WebAssembly, but you could see web application frameworks being built in any language. You could have Java apps that are compiled down to WebSssembly, Rust. Really, you could write any language in any framework and that could turn into WebAssembly. It provides so many different options for how to build apps, because it's this kind of this native target.

[00:54:22] JM: All right. Well, that will make your job much harder.

[00:54:23] MA: Yes, it will. It will.

[00:54:25] JM: Matt, thanks for coming on the show. Been great talking.

[00:54:27] MA: Thanks, Jeff.

[END OF INTERVIEW]

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]

