EPISODE 1126

[INTRODUCTION]

[00:00:00] JM: WebAssembly allows for the execution of languages other than JavaScript in a browser-based environment. But WebAssemly is not widely used outside of a few particular niches such as Dropbox and Figma. Nicolo Davis works on an application called Boardgame Lab, and he joins the show to explain why WebAssembly can be useful even for a simple application like Boardgame Lab. Nicolo also shares his reflections of Typescript, Rust and the future of web development. He talks through the client server interactions, performance, error handling and the process of an actual migration that he made. It's an interesting case study in WebAssembly. I hope you enjoy it.

[INTERVIEW]

[00:00:44] JM: Nicolo Davis, welcome to the show.

[00:00:46] ND: Jeff, thanks for having me.

[00:00:48] JM: You work on something called Boardgame Lab, and we're going to talk about Boardgame Lab, but really it almost doesn't even matter what app we're talking about here. It's just a stand-in for talking about WebAssembly, and Rust, and the migration that you did. But just because we need to have some kind of prototypical example, let's talk about Boardgame Lab. What is that app that you have created?

[00:01:10] ND: Sure. Boardgame Lab is a web app that allows you to prototype and play test board games and card games. It's something that allows you to quickly iterate on your design. It comes with a built-in layout editor where you can edit graphics on the browser itself. And then from that, you can create a full-fledged game that you can play with people over the internet.

[00:01:33] JM: Tell me more about the spec for the game. What have you needed to build in order to implement it?

[00:01:37] ND: So if you mean the tech stack, the frontend is mostly Typescript and Svelte, and I've converted some of the state logic to Rust and WebAssembly, and then the backend is all Rust.

[00:01:49] JM: Let's talk about the spec in terms of what exactly have you had to build in order to make the game. What have you had to engineer it up from a high-level?

[00:01:57] ND: To start out, when you first open up the app, it asks you to create a layout for the different components that you have. So for example, on a card, you might want to design a card with a title and some text in the body. And once you do that, it allows you to connect that layout to data from a spreadsheet. For example, you could spin-up a hundred cards from the same layout. And once you have that, then you move on to the next phase of play testing. And it's a single click to just go online. And you can play with people that you share the link with. And then the platform aggregates all of the statistics and gives you a little report at the end.

[00:02:37] JM: What's the client server interaction for Boardgame Lab?

[00:02:40] ND: So it's a single-page app. Basically, whatever is running on the client is able to understand all of the state upgrades. It contains all of the game logic essentially. The server acts as the single source of truth. So basically when a client makes a move, the client updates itself and then it sends a message to the server saying that, "Hey, I moved this card from here to here." And then the server updates its copy and then relays that message back to all of the other clients. So that's kind of how everybody stays in sync in real-time.

[00:03:13] JM: Now, that doesn't sound like a very performance-intensive application. You got pretty straightforward client server interactions. What's the complexity there? Why would you ever need to do some kind of complex migration for this?

[00:03:29] ND: Right. So the complexity comes from the face that I also want to use bots for Boardgame Lab, and the algorithm that is uses, Monte Carlo Tree Search. Any sort of a tree search, which explores the game tree can be quite computationally heavy. And for that, I felt like it would make sense to speed up the state updates. And that's one of the reasons that I chose to migrate some of the typescript code to Rust and WebAssembly.

[00:03:59] JM: Let's talk a little bit more about that. Monte Carlo Tree Search, so where is that performance intensive? Or how is that performance intensive?

[00:04:06] ND: Okay. So let me give you an example. So let's say you have a card game and you have a hand of cards, let's say 5 cards. Now, from the perspective of a bot, it could play any one of those five cards, right? And so Monte Carlo Tree Search, the way it works, it explores the game tree, which means it might try all five options and then look at the ramifications of each one of those. And it keeps going further, further, further down the game tree. So like you play this card and then the opponent might play something else. And it keeps exploring that game tree. But the difference between a total brute force search of the entire search space, it uses statistics at each step to determine what's the right tradeoff between exploiting a particular versus covering more breadth.

So if you have a large search space, a lot of card games tend to have several options at any given point of time. Then you need to execute a lot of code effectively. So several thousand iterations to come up with sufficient confidence that a particular move is good. So that's what's computationally expensive about searching game trees.

[00:05:18] JM: Can you tell me more about the client server interaction, like just so we really iron this out and make it apparent to the listeners?

[00:05:26] ND: Sure. From the perspective of a multiplayer game, when the client makes a move, right? Let's say you're playing a single player game with one bot like all running on the client side, you can make a move and then you have the bot that can take a couple of seconds to play, right? And then all of that thing runs on your browser. Now, if you shift gears a little bit and then you move to a multiplayer game with two players and a bot, now you have a client, a human player that makes a move. And then the human player sends an action to the server, right? And the server now relays that action to the other client that's in the game. And the server also controls the bot. So now the bot is now running on the server. So these are the two different scenarios.

[00:06:10] JM: Okay. The original architecture was an isomorphic architecture completely in JavaScript. Explain why that was useful? Why is an isomorphic architecture useful?

[00:06:22] ND: Sure. Yeah. An isomorphic architecture – And by that, I mean, having the same code available on both the server and the client. It is useful for apps like this, because you can have the client update itself independent of the server. For example, if I move a card from location A to B, the client knows how to do that without having to wait for a response from the server. So even though we use the server as the single source of truth, and in case any of the client goes out of sync, you actually fallback on the state of the server. But the client is still able to optimistically make an update without having to wait for the network round trip. So that's the advantage of an isomorphic architecture.

[00:07:05] JM: Is JavaScript a good fit for this kind of app? Or why is JavaScript not a good fit for this kind of app?

[00:07:14] ND: Well, absolutely. Yeah. JavaScript lends itself very well to isomorphic apps, because you can run the same code on both the client and the server. So on the server, you can use Node.js. And JavaScript is plenty fast. So most apps shouldn't really need to worry about performance until they're doing something computationally expensive, like image compression or running Monte Carlo Tree Search. So the approach with JavaScript is you push client-side code to the server effective. And with Rust and WebAssemly, you're doing the opposite. You're pushing server-side code to the client.

[00:07:51] JM: Okay. WebAssembly is where we need to get to. How is WebAssembly useful for this kind of application potentially?

[00:07:59] ND: WebAssembly is a relatively new standard that it's a binary format that can be executed by the browser. And there are several languages that target WebAssembly, Rust being one of them. But you could write code in C++ or even Go for that matter to target WebAssembly. The Rust toolchain is particularly mature for WebAssembly. So you could write Rust code and just compile it to WebAssembly. And then all of the tooling handles the interop pretty smooth. So what that gives you is the ability to write code that you would traditionally run on the server, but have it available and runnable on the browser as well.

[00:08:40] JM: This migration from JavaScript to WebAssembly, has anyone else done this kind of migration before?

[00:08:47] ND: I'm sure people have. I haven't read many experiences of people doing that. I guess maybe the most high-profile that I've seen is probably Figma. I know Figma migrated a bunch of stuff to WebAssembly, and I think one of the – I mean, the performance benefits aside, I think they also highlighted the fact that WebAssembly modules can also be smaller and faster to load on the browser. So that's one of the things that I remember reading from their blog post.

[00:09:18] JM: And was there an easy path to doing this kind of migration?

[00:09:23] ND: Yes. So let me describe the code that migrated. So, in the card game for example, most of the manipulation and the state updates are just moving things from one data structure to another data structure. So let's say you have two decks, A and B. If you move a card from A to B, it's effectively disappearing from A and then appearing at B. Now, these kind of state updates, of course while individually they're not computationally expensive, but then when you're doing a game tree search and you're running them thousands of times, of course, then it can start to add up. But from the perspective of writing something like this in JavaScript or Rust, it's a fairly straightforward migration, I would say. And I think it's all about making sure that the tool chain is mature enough to support the interop. The interop is where most of the problem lies.

[00:10:13] JM: So, what's important to note is that WebAssembly is just a lower level, essentially, compilation format for other programming languages. So it's up to you what higher-level programming language you want to use. So how do you select between what programming language to use at a higher level?

[00:10:29] ND: Right. Coming from JavaScript, obviously, a natural choice might be AssemblyScript. AssemblyScript is a stricter version of TypeScript that targets WebAssembly. I did look at TypeScript, but I think at the moment, it has several restrictions. For example, you can't use closures. For me, that was a bit of a non-started, because I felt like it would be a significant migration to move to AssemblyScript. When I looked at Rust, one of the – This is maybe my personal advice, but I came from a background of C++. So it was a language that felt very natural to me. But I felt like for the handling of data structures and all of these manipulations, I felt like the stricter typing of Rust was particularly helpful for me. So, migrating the codebase to Rust not just enabled the use of WebAssembly, but I think it also cleaned up my codebase quite a bit.

[00:11:28] JM: So, can you describe the interaction between Rust and WebAssembly in more detail?

[00:11:32] ND: Yeah, sure. Like I said, the tool chain on Rust is fairly mature for WebAssembly. So, I can just write, take a Rust module and basically just run it through a tool called wasm-pack, and it handles all of the compilation to WebAssembly. And then on the JavaScript side, there are plugins available for different module bundlers like Webpack and Rollup. I use Rollup. And it's as simple as importing your Rust module and it will create a JavaScript file that you can import as an ES module.

So from the perspective of JavaScript land, you are effectively just importing another JavaScript module and just using the functionality inside, which was originally written in Rust. But from the perspective of your app, it doesn't really matter.

[00:12:26] JM: Tell me more about the Rust and WebAssembly ecosystem and that maturity of it.

[00:12:32] ND: Yes. There are definitely some pain points at the moment. I wouldn't say it's perfect. When you do cross the JavaScript WebAssembly boundary, that you do pay a cost. So, for example, the way Rust works right now, you have to serialize your JavaScript data structures to JSON and then de-serialize it on the Rust side into Rust structs in order to use this more complex data types. Because WebAssembly, the spec at the moment supports very primitive data types. So you can only pass it on integers. This is changing by the way. So there are proposals in place to make this easier to work with. But at the moment, all of the tools that handle the interop between WebAssembly and JavaScript need to do some fancy things to facilitate the parsing of higher level data structures.

So you do pay a cost when you cross that boundary. So I would encourage anybody that's thinking about using WebAssembly to actively profile and benchmark their code to figure out if the performance gains are worth it or if they will even matter. That said, I think WebAssembly is moving pretty fast. I think there are a lot of proposals in place to speed up different parts of the ecosystem. And then of course there is browser support to be taken under consideration.

Firefox and Chrome has been pretty aggressive in their adaption of WebAssembly. And at the moment I would say it's a good base system to work off, and that can only get faster.

[00:14:06] JM: How does the experience of writing wasm compare to writing JavaScript?

[00:14:11] ND: Well, yeah. That really depends on the language that you're using to target WebAssembly. So from my perspective, I would say using Rust feels like a more natural fit for that particular part of my application. That said, I wouldn't use Rust for all of the frontend interactions, for example. It is possible to do that. There are people that have written frameworks similar to React, for example, that you could use in Rust to create an app that uses WebAssembly all the way. I'm more of use the right tool for the right job kind of guy. And I prefer to use Rust and WebAssembly for the computationally heavy parts of my app and then use a more traditional JavaScript stack for the rest.

[00:14:56] JM: Can you tell me more about what the tool chain is to build this kind of client server interaction with Rust?

[00:15:03] ND: Yeah. You use Rust to basically create essentially opaque functions that I call from JavaScript. For example, in the context of a card game, you might have an update state function that basically just takes the existing of the game board and then add action. An action might be move a card from A to B and then it basically returns to you the new state of the game board, right? On the Rust side, this is just a simple function that you can export. And then wasm-pack and then the Rollup plugin that I alluded to basically convert this into a JavaScript module that you can import like any other JavaScript module on the client.

And then on the client, you could just use it as a regular JavaScript function essentially. So, it's a standard SVAA written in Svelte. And when the user clicks on a card or drags it location A to location B, you basically just invoke this function that updates the data structures that hold the state of the board. So that's kind of how the entire stack works.

[00:16:12] JM: All right. Well, now that we've given an overview for what you did hear, let's talk about the changes. How does this migration to Rust and WebAssembly actually changed what you've built?

[00:16:24] ND: Yeah, that's an interesting question, because I think there are definitely some things that I have done a bit differently after moving to Rust. So Rust has really powerful Enums. So you can have Enums that also hold payloads. And this is a particularly powerful way of modeling data. And I think it has changed the way in which I have modeled the actions that I send over to the server. Until then, my client-side portion of the app was – I guess it would be familiar to anybody that used Redux or any library like that. Basically, you just have some state. And in order to update that state, you have to perform an action and you send that action. And that's the only way you're allowed to update that state. Now, that stays the same. But now the actions I think can be modeled slightly better. I think they have richer payloads that I'm able to handle more seamlessly using the code in Rust that I would have to do a bit differently in Typescript.

[00:17:26] JM: How does error handling change?

[00:17:29] ND: Yeah. So, Rust has – I would say one of the better error handling mechanisms that I have come across in the programming language, it makes it really, really difficult to ignore a code path that could result in an error. So, for example, if you're performing any sort of operation on a data structure, indexing a list or deleting something. Any code path that could result in an error usually gets passed around in a result object that could be either successful or contain an error. And you have to explicitly opt-in to a code path that can result in a crash. Rust has really powerful pattern matching. So the moment you introduce a code path that could result in an error, it will force you to handle it or basically make you explicitly opt-out of it. So I feel like it's very easy to keep track of everything that's happening in your code path. And it makes it fairly robust on the server-side. So I have very great confidence that if my code compiles, it's not really going to crash. So that's been quite a blessing after moving to Rust.

[00:18:43] JM: How does performance change?

[00:18:46] ND: Performance, at least for me in the current state of my app, I'm still in a pre-beta phase. And I'm not particularly concerned about serving a high amount of traffic at the moment. But that being said, I will say that for a solo dev managing a project like this, performance does matter even in the early stages, because the Rust server that I'm using for handling the multiple traffic uses a lot less memory than the equivalent Node.js server that I was using previously. And this allowed me to use the same DigitalOcean droplet that I'm using right now for a longer period of time before I need to scale out. So I think that considerations for performance at this stage are lower utilization of CPU and memory rather than having to handle massive amounts of traffic.

[00:19:41] JM: So, tell me more about why you did this. Let's just refresh people. Why did you migrate to Rust and WebAssembly?

[00:19:51] ND: Okay. The primary motivation was to facilitate the ability to run bots into platforms. I have some previous experience writing Monte Carlo Tree Search bots in JavaScript through various open source libraries and so on. And Monte Carlo Tree Search can be computationally quite expensive depending on the complexity of your game. And if you have a game that has many different choices at every point, the branching factor quickly adds up and then you can have a bot that takes several seconds before it can make a reasonable move.

That said, Monte Carlo Tree Search is a very elegant algorithm and allows you to bail out early. So if you only have a limited amount of time for it to run, let's say, two seconds, you could run it for two seconds and then just see what it does. But if you are able to run more iterations in those two seconds, you're going to get a bot that plays more reasonable moves. That was the primary motivation to facilitate faster state updates so that you can run more iterations for the bot to be able to churn out its logic before it makes a move.

That said, after I did start the migration, I started to notice a few other benefits from the stricter typing in Rust and better error handling and so on. And the WebAssembly interop has been pretty smoth to the point where I haven't really noticed any sort of friction. I use Rust for the state updates. I use JavaScript for frontend animations and so on, and it seems to work just fine.

[00:21:21] JM: Tell me more about how the client server interaction has changed.

[00:21:25] ND: The client server interactions haven't changed all that much. The only difference being that, now, instead of running a Node.js server on the server, I'm running a Rust server. But from the perspective of the client, it's still the same. It still makes a web socket call to the server to say that, "Hey, I moved this card from here to here." And then the server basically relays that to the other clients. From the perspective of the client, the only thing that's different is now it's importing a WebAssembly module and calling that to performance data update rather than using JavaScript code, and that's the only difference.

[00:22:02] JM: So, how has this been advantageous to you?

[00:22:07] ND: The primary benefits so far for me have been the performance benefits, and also, like I said, the stricter typing in Rust has helped me to clean up my codebase quite a bit. Typescript's typing system is quite expressive, and I would say in many ways even richer than Rust's in terms of the ability to express conditional types and so on. But one of the problems with TypeScript is that I would say it's more of a type hinter than a type checker. It's more of a compiled time dev tool I would say, but it doesn't really have a runtime component that's able to validate your data structures and the data that's moving through your system. Rust is a lot more stricter than that. For example, you can't de-serialize data into a Rust struct that does not fit exactly those data types that you have defined for it. So you get some amount of validation out-of-the-box for free. And so there are benefits in using a programming language that has stricter typing for a piece of code that manipulates a lot of data structures.

[00:23:11] JM: Do you have any broader reflections on programming languages from making this migration?

[00:23:19] ND: Well, I would say I think it's important to identify parts of your application that could benefit from using a different tool and then explore that. Like I alluded to earlier, I think there is a temptation to use the same infrastructure and ecosystem everywhere, and there is merit to that for sure. I think there's definitely benefit to using the same programming language on both the server and the client. But I think if you can identify an area that can really benefit from using something a bit more specialized, I think there are a lot of benefits to be had. And especially if you are comfortable writing code in multiple programming languages and you're comfortable switching between the two, I think that's the key point here. I think it's important to isolate these bits enough that switching between the two doesn't feel jarring. So like my experience, I'm in Rust mode when I'm updating data structures, and I'm in JavaScript Svelte mode when I'm performing client-side animations, and so on. It doesn't feel that jarring for me. And so I think that's the key insight to kind of figure out if there's a good way to bucket these two areas and then focus on each independently.

[00:24:35] JM: Do you have any reflections on Rust and Typescript specifically?

[00:24:41] ND: I would say that from a frontend perspective, Typescript is probably the first tool that you should reach for if you're writing any sort of application. And I think Rust should be something that you only consider if, A, you're looking for better performance or something that you feel like could organize a codebase a bit better because of the better error handling and richer Enums and so on. The ecosystem moves very fast as you well know. So I'm not sure what this tradeoff is going to look like in 5 years from now. But it's an exciting time to be working with Rust and WebAssembly, because I think I can use the same codebase and be fairly confident that it's going to get faster as different browser adapt all of the different improvements that are on the table right now.

[00:25:31] JM: Tell me about broader predictions for web development and how this migration has shaped your projections about web development in the future.

[00:25:41] ND: Yeah, that's tough question. I think if we look back, I think we started out with server-rendered apps, right? Servers that just spit out HTLM and then the browser could sprinkle some – I mean, you could sprinkle some JavaScript on there and then the browser could bring that app to life on the client-side I think we're kind of gone the opposite direction from there to very, very thick clients now which have a lot of JavaScript. And now your server is basically adjust your API or data layer.

I think the evolution has been moving towards more isomorphic apps. I do see a lot of adaption on Typescript in particular, I think, that allows these kind of apps to be built where you have a lot of logic on both the server and the client that can be shared through one codebase. And now with the ability to use WebAssembly, I think we're going to see more of that, because now you can use the programming language of your choice. So you don't like to write JavaScript and you could use C++ instead if you chose to.

I think we're going to see more of that. And WebAssembly at the moment is not a great fit for actually manipulating things on the DOM, which is why I don't use it for that. But that bit of interop is also going to get faster. So I think we might start to see people use WebAssembly to also do more client-side stuff. So it'll be interesting to see how that ecosystem evolves.

[00:27:11] JM: What are the shortcomings of the WebAssembly ecosystem right now?

[00:27:16] ND: Yeah. The primary shortcoming at the moment is the cost that you pay whenever you cross the JavaScript WebAssembly boundary. Firefox has been doing a lot of great work on this frontend. They write a lot of detailed blog posts on how they've made it faster in Firefox. And then the other browsers are following soon.

The primary cost that you pay right now from a perspective of Rust and WebAssembly is that whenever you pass in a complex data structure that's not an integer or string, you have to serialize it to JSON and then de-serialize it on the WebAssembly side. So that cost is something that needs to be taken into consideration. It could cost you 50 milliseconds.

From the perspective of my app, it doesn't really make a difference, because I'm handing it something that's computationally expensive. So I basically handoff to WebAssembly, "Hey, go and churn on this. give me a good move after you've run this thousandth of iteration." So that cost doesn't really matter too much to me. But if you're writing an app that uses WebAssembly that's doing something fairly light, you might want to take that interop overhead into consideration. I think the important point is to always profile your app and make sure that you have a demonstrable performance benefit before you migrate over.

[00:28:30] JM: Was this migration that you made really necessary? Did it really improve things enough to be worth it?

[00:28:36] ND: I would say yes. Yeah. I think I find that the Rust codebase is a lot easier to work with for the state updates that I'm performing. I think with Typescript, you often do need the help of an immutable library helper like Immer, something else to kind of facilitate these kind of data structure updates. I think with Rust, a lot of these feels a lot more natural. I think especially with the good error handling, I think I feel like it's improved my codebase quite a bit.

I don't regret to change irrespective of the performance implication. I think I would have been quite happy with the migration even if it was performance-neutral in a way. But I think it's also a nice benefit that the WebAssembly is faster for my application.

[00:29:22] JM: What would have happened if you would have just kept building your app without migrating?

[00:29:27] ND: Yeah. So I think if I chose to stay with Typescript, I probably would have had to explore performance optimizations in JavaScript land. And I think you can have a look at [inaudible 00:29:40] passing around a lot of things and simple arrays, watering down your data structures. And it's definitely doable. But I think the cost over there is it does decrease the readability of your codebase. So I think with this migration, I think I can have a codebase that's very readable and performant at the same time.

[00:30:00] JM: What are the other use cases where a migration like this might be useful?

[00:30:04] ND: I think one of the popular use cases for WebAssembly right now is any sort of image processing that you might want to do on the client side. So let's say you have an image upload field and then you want to do all sorts of transform at the image on the user's browser before you send it over to the server, I think WebAssembly is a great fit for that. And there are a number of libraries available to do that.

I think that's one area to look at. But basically, the rule of thumb I think whenever you have anything that's computationally expensive, I think WebAssembly is a good choice and it's definitely worth exploring.

[00:30:41] JM: Tell me more about what examples you've seen of WebAssembly in the wild. I remember, I've seen like Dropbox uses it, Figma uses it, but I don't know of any other in-the-wild examples.

[00:30:54] ND: Yeah. These are the two high-profile ones. I have occasionally seen people post on Hacker News about apps that they've migrated over to WebAssembly. And I think a lot of the examples that I've seen actually have involved actually converting the entire app to WebAssembly, so where basically they use something like Rust for even the DOM manipulations. I'm not a big fan of that approach actually I much rather prefer the approach of identifying a portion of your app that could use rewriting in a different language and then migrating only that big. And I think Figma is a good example, and I think they have a pretty detailed blog post about their migration and how it has sped up their app.

[00:31:38] JM: Well, what else do you think your app would benefit from if you could write more Rust code? If you wrote more low-level code?

[00:31:47] ND: So let me clarify that for a minute though, because I think I wouldn't really classify Rust as low-level code here, because you can write fairly high-level abstractions in Rust just like you would in Typescript. And that hasn't changed. So when I'm in Rust land, I'm not really particularly bothered about the placement of different data structures and how they're in-memory representations look like and so on. But I would say that when you are writing a code in a different programing language like Rust, you have to really exploit the rich standard library that it comes with. And as long as you are using the happy part of there of fitting all of your data models into existing solutions, then I think you'll have a great time writing in a language like Rust.

[00:32:37] JM: Did this project give you any other reflections on Typescript, or web development, or WebAssembly? Is there anything else we should discuss?

[00:32:46] ND: I think the big takeaway for me was that you're not restricted to using JavaScript for everything, and I think it gives you more options, and that's a great thing. And I think that's what I would encourage people to explore. I think people have, in the past, tried to use things like mscript and asm.js and so on to get encored on the browser that's not just JavaScript. But I think now with WebAssembly, you have an open standard, which means there's going to be better support across the entire browser stack. And you're going to have different languages all develop customized tool chains that can target with some WebAssembly. So I think we're going to have a much richer ecosystem. That's my takeaway.

[00:33:32] JM: Any other predictions?

[00:33:34] ND: No. I don't think so.

[00:33:37] JM: Well, I thought this was a really good use case of a migration of an entire app to WebAssembly. Are there any recommendations you would give or lessons learned for people who might be attempting a WebAssembly migration in the near future?

[00:33:53] ND: Yeah. I think the important point is to profile your app. I think it's nice to maybe take a very small section, migrate a little bit of code and just do a quick litmus test to see if this is something that makes sense, because there are some wrinkles in the ecosystem. It's not all perfect, and it might very well end up being the case that things turn out for the worst if you do a migration like this. I think it's important to make sure that you have the right kind of monitoring in place, the right kind of benchmarks that you can test against. If you do identify that this is something that is beneficial to your app, then I think the other consideration now is what kind of language do you want to use? Do you want to use Rust? C++? Something else. And once you're comfortable and productive in a language like that? Then I think it becomes a very smooth process of writing a certain part of your app in Rust and then the rest of JavaScript.

[00:34:49] JM: Cool. Well, Nicolo, thank you so much for coming on the show. It's been real pleasure talking to you.

[00:34:53] ND: Thanks a lot, Jeff.

[END]