EPISODE 1313

[INTRODUCTION]

[00:00:00] JM: The term boilerplate code refers to code sections that are repeated across many projects with little to no variation. Every developer is familiar with boilerplate code, whether it'd be palm.xml files in Java, or setting up ReactJS applications. Tweaking boilerplate code for every project is inevitable. Actually, the company Wasp believes writing boilerplate code doesn't have to be part of web app development. Wasp is an open source declarative DSL for developers who want modern frontend, backend and deployment experiences without writing boilerplate code. With Wasp, you can describe high level features with Wasp DSL and write the rest of your logic in React, Node.js and Prisma. In this episode, we talk to Matija Šošić, CEO and cofounder at Wasp.

[INTERVIEW]

[00:00:48] JM: Matija, welcome to the show.

[00:00:50] MS: Thank you, Jeff, for the invitation. Happy to be here.

[00:00:52] JM: I'm thrilled to be talking to you. I have done a few shows about opinionated full stack web development platforms. And from what I can tell, you have a novel take on this. So you are building Wasp, which is an open source declarative domain-specific language for using modern web development tools. So you basically are saying, "Look, the way to build web applications, we're going to be opinionated, is React, Node.js, Prisma, etc." Tell me why you have formed these opinions about these technologies being the standards for web development?

[00:01:37] MS: Yeah, it's a very good question. And I would say it is true that we're being very opinionated, but not so much in the sense of the technologies that we are allowing people to use like React and Node.js. It is more about the abstractions that we are offering to a language. So maybe the main difference between Wasp and the typical frameworks like Ruby on Rails, and MeteorJS is very good example, because they also cover full stack, for example. So I would

say we are now covering the same problem, which is building web applications. The difference in us, in them is in the technical approach, which means on the one side, you have typical web frameworks, which are based on libraries and runtimes. And on the other hand, Wasp is going with the language approach, as you mentioned, domain-specific language.

And actually, the main reason for this technical decision was because of the reason that we have higher level and possibilities to make more and higher abstractions. So right now, we're supporting React and Node.js. But actually, the big goal with Wasp is to be able to support multiple technologies and also multiple architectures in the future. So right now, it's React to Node.js But we also want to add support in the future for a Vue, for example, and even more languages on the backend. For example, Go, or Ruby even, whatever proves to be a good candidate for that. So I would say that is the main difference, and how we are opinionated.

[00:02:54] JM: Alright. So let's go through a few prior case studies that we've explored. So one that comes to mind is RedwoodJS. I imagine you've seen that, right?

[00:03:06] MS: Yes, of course.

[00:03:07] JM: So RedwoodJS, we did a show with the – I think the original founder of GitHub. Tom Preston-Warner is heavily involved in RedwoodJS. So that takes an opinionated approach to web development. Can you use that as a data point of comparison?

[00:03:24] MS: Yes. Yes, exactly. I'm happy to do so. So I would say for that, very good examples of modern and up and coming frameworks are Redwood that you mentioned. Another one is also Blitz.js. And I would say the main thing with them is that they're both also full stack frameworks, which means they are covering both frontend and backend, which is also what is Wasp doing. Especially, since we have now – We have Ruby on Rails, everything started back then. And when everything was on the server side, everything worked perfectly, and Ruby on Rails was a perfect solution. But then **[inaudible 00:03:55]** in the sense of the technology. And we got these be division on the frontend side, which become much more heavy, and much more client-oriented. And so it became more as a data storage, which will be just serving data to the frontend.

So I would say with this evolution, we actually got a big gap between frontend and backend. And there was much more need for solutions to manage the state and do it independently. And there was a lot of new ways and a lot of new necessities introduced enable to inform the developer to be able to build a full stack reapplication. So I would say, again, after the technology has evolved a lot, and we are starting to see some patterns in how full stack web apps are being built. Now we feel, actually, there is a new trend towards, let's say, the next generation of Ruby on Rails, which is actually supporting that kind of development with all the best practices, opinionated, but supporting the modern full stack architecture and environment.

So I would say in that sense, we're going after the same trend. And yeah, basically the main difference between Wasp and typical frameworks like Redwood and Blitz is the technical approach, which means Wasp is actually a language. So we are not tied to the specific language like JavaScript, or specific architecture. Also, actually being a language, we can provide, let's say, whatever experience we want. So we can even reduce the amount of the boilerplate and make it even, let's say, more human readable to the developers that are using it.

[00:05:21] JM: Okay, so that's really interesting. Because the human readability, that hints at a developer ergonomics effort. Tell me about your – Actually, I'd love to know a little bit about your history and how you got to working on this.

[00:05:35] MS: Yeah, sure. I think also, ergonomics is a very good word. And this is what we are definitely going after. But yeah, let me first share a bit about the founding team and what was our background. So the creators of Wasp, it's me and my twin brother, Martin. So just shortly about us, we are both computer science engineers. We graduated our master's. And during University, we worked in companies like Google and Palantir, doing our internships there. We have always been much more, let's say, on the algorithm and the backend side, especially on the bioinformatics and machine learning. For example, Martin actually built one of the – Now a bit more well-known bioinformatics libraries for DNA sequence alignment, which is now being used by companies like Bigbio and Similar. So we have always kind of been – We never thought of ourselves like web developers. It was more like general software developers, and we just used whatever we needed to use. But as we kept building projects and doing different stuff, we always had the need to build web application for whatever algorithm we built. And we actually wanted to make it available to developers to use it.

And funnily enough, often, this web application part proved to be the more complex than actually algorithm or machine learning thing that we have built initially. So I would say from this experience, we have been building web apps starting from jQuery, and Angular, and Backbone, and now React. And also from the backend, we went to everything from Java, Python, and now Node. So we always felt moving from one project to another, we always had to relearn the latest stack and all the best practices on how to build a web app just let's say to build a similar web application again and all over again.

So I would say for us, this was one of the big motivations why we wanted to introduce, let's say, a language abstract enough that it can capture the essence and requirements of the web application, and that is not going to be changing so much is the underlying state is changing, because if you're frustrated with needing to relearn all the best practices just to build the same application all over again.

[00:07:43] JM: Can you tell me a little bit more about those points of friction that you encountered in developing web apps over and over again? I assume there are some things that you figured out, if you streamlined these things, your web development process would go a lot more smooth.

[00:08:01] MS: Yes, definitely. I would say there is multiple things that we have encountered so far. And I believe there are more that we don't have a good understanding yet. But we will get into the future. But for example, building a typical crud application. So there is always – We have been building mostly single page applications, which means having a client, a frontend, and also a dedicated backend server for the data. So I would say, especially the one thing is just to learn the latest framework, for example, is it backbone, or Angular, or react? So this is what takes a certain time. And then on the other hand, we have also being – Let's say, typically, it is authentication and communication between the server and the backend, pages, routing. There is always a set of best practices on how to do and build stuff around that, especially with focus on the security and performance. So I would say these are one of the most common things that we believe are the patterns right now in the application development. So this is what we are targeting with Wasp right now initially.

[00:09:01] JM: Gotcha. Another data point., do you know Dark Lang?

[00:09:06] MS: Yes, of course.

[00:09:07] JM: All right. So Dark Lang, really interesting company, basically trying to shoot for the moon. Trying to unify a language with a hosting model with, I think, development systems. Led by Paul Biggar, founder of CircleCI. I think it's a unicorn company. So really ambitious project with a credible team. When I was talking to them, I got the impression they bit off more than they can chew. That said, I'm a complete believer. And I would have invested in the company if I could have. How does your ideology compared to Dark Lang's?

[00:09:44] MS: So I would say there are some similarities, and there are also few key differences in what I understand in their approach. So from what I know about Dark Lang, it was mostly focused on the backend, actually distributed backend. Actually, Wasp is more like full stack but with frontend and backend. But maybe the main difference with Dark, it was in my understanding a general programming language. So they actually built a new language from scratch.

So the main difference with Wasp is that we are not trying to do that. Wasp is not a general programming language. I mentioned in the beginning, a domain specific language. And we are not trying to make a language which is going to be powering both frontend and backend at the same time. Wasp just a high-level language, which is used for some high-level web application abstractions. And then we still rely very much on developers plugging in their own pieces of code right now in React and Node.js.

So I would say that also allowed us to move the initial version of Wasp pretty fast and also now to move faster, because we are not trying to build the whole solution from scratch. And we are very much focusing on the interoperability with the existing stack. So in our opinion, this is the – Let's say this is our unique insight in the future. We believe that is very important.

[00:10:57] JM: All right. So we're easing into the conversation. You're working on a full stack web development platform with an emphasis on open source defaults. Before we get into that business model, are you thinking this is going to be a unified hosting and frontend framework?

[00:11:17] MS: Yeah. I mean, Wasp is not the frontend framework. It is a full stack framework.

[00:11:20] JM: I'm sorry. Full stack framework.

[00:11:21] MS: I mean, maybe go to the framework. It's maybe not the good word about the full stack solution. So I think that is definitely the good wording. But yeah, I will say there is multiple potential possible monetization. And we are still a bit far away from that, because we had just started, and right now being in alpha. But obviously, we have thought about it. So I think there are several possible paths to do it.

On one hand, what you said, which I think is pretty straightforward. Also, looking at other solutions, is basically having a hosting platform. Allowing developers to easily deploy and host their applications. And I would say on the other side, we should also be logical, is introducing, let's say, more answers, which can actually be part of the enterprise license.

[00:12:03] JM: Got it. And how do you build enough of a following to make that a viable business?

[00:12:13] MS: Yeah, it's a good question. I would say that is actually our main focus right now. It is not so much about figuring out who is going to be the first customer. It is just focusing on building something that developers love and want to use. So for that, I can share what has worked for us so far and how we are thinking about it in the future. But it is very much a work in progress. So I would say we started working at Wasp full time about a year ago, now almost a year and a half. And for the majority of the time, it was actually just me and Martin trying out a prototype, building out different abstractions, seeing what works, what doesn't. But we have also been resistant to putting it in front of developers, because we wanted to make sure we are not building this in isolation. We wanted to make sure we are getting input from other people who are very knowledgeable in the field.

So they've guided us and definitely influenced a lot of what we're building. And I would say we use the scope in some ways, we extended in others. But with all that information, we kept building, and we released an alpha version in December last year. So with that, we started with

some good traction on **[inaudible 00:13:19]**. So Wasp was number one **[inaudible 00:13:20]** that day. So they gave us an initial boost. And we started seeing a lot of people using it for hackathons, **[inaudible 00:13:27]**, or just finding it out and building some projects, joining our open source community.

After that, Reddit has also proven to be a great source for developer engagement. Even not just the **[inaudible 00:13:39]** and even more like SVGO, because we can share a lot of insights. And what are we building? Ask for advice. So we have been one of the – Maybe even the most voted posts in several Reddit communities like ReactJS and JavaScript. So that gave us a lot of feedback and boost. And with that, eventually we entered YC, which was also obviously great for kind of maybe more strategizing, big vision. How do we want to focus on this? And with that, we also had a very successful Hacker News launch. So wide Hacker News launch was I would say one of the most successful launches in the whole batch that we were a part of. So they've helped us a lot in visibility, and also very high-quality feedback from developers.

[00:14:21] JM: Now that we've set the stage, let's talk a little bit more about the technology and the vision. So, today, I'm actually getting started with a new company that I'm making. I'm making this gaming company. And I'm doing a full stack JavaScript application. I've got my own big vision for the game. It's obviously going to be React application. I kind of want GraphQL involved, but don't really want to deal with GraphQL from day one, because of setting up a GraphQL server. But I'd love to have the easy optionality later on. I want other sane defaults. I don't know, linting, stuff like that. Obviously, there's going to be Auth. I see what you're doing. And it seems quite useful.

A quote from your website, "Wasp aims to be at least as flexible as the traditional web frameworks like Ruby on Rails." I mean, is it so much to ask to have a full stack JavaScript framework that is as good as Ruby on Rails? It seems like the world should have asymptoted there by now. And I think Next.js is that. So why – I guess that's another point of comparison. Can you tell me about comparing to Next.js?

[00:15:28] MS: Yes, sure. So I would say maybe Next.js and Blitz.js, which is basically I would say the key difference is Next.js is a bit more oriented towards frontend. And Blitz.js evolved this natural kind of – Natural product, which wraps Next.js and its full stack capabilities around it. So

I would say if we are comparing, let's say, Blitz.js is closer to Wasp, because they are like strictly focusing on the full stack they experience.

[00:15:55] JM: Can you give me more context? I haven't worked with either of these frameworks. Or, sorry. Sorry. Development platforms. Whatever –

[00:16:01] MS: I would say the main difference, for example, Next.js doesn't come with database. So you can run some serverless functions and stuff like that, but you don't have a database. And basically, without that, you are missing actually a lot of these things to do your full stack application. So basically, Blitz.js is that kind of an upgrade, giving you a database, and then all the stuff that you need around it to make things work. I would say that is the main difference between Next and Wasp. Wasp is a full stack solution covering frontend, backend and database altogether.

[00:16:32] JM: And so your databases is what? Mongo? Postgres?

[00:16:37] MS: So right now, we're using Prisma as a database abstraction. And Prisma is supporting Postgres right now. But they will also soon add support for Mongo. So altogether, we are very happy with using Prisma as the database abstraction, because they also use DSL for their data modeling. So it fits very well into our overall premise.

[00:16:56] JM: Okay. Shout out to Prisma. I haven't talked to them in a while. They have been trying to make databases better for developers. What has Prisma done to make databases better?

[00:17:08] MS: I would say it's in many ways similar to our original with Wasp, only that they're focusing on database part, and we are focusing on like full application part. So let's say they also introduced their own DSL, which is basically abstracting databases. So you can be either using Postgres or Mongo. And that is exactly what we want to do with Wasp, Making people to use whatever technology they want while still having all the same defaults and all the best practices in place. So Prisma is also doing that in their own way. But yeah, focusing on database layer.

[00:17:43] JM: So if I remember, Prima is like middleware between your frontend and your database layer. That makes it easier to query. It's like an ORM, but not an ORM. I always kind of forget what exactly they're doing. Can you remind me?

[00:17:59] MS: Yeah, there's also differences between Prisma one and Prisma two. So now it has changed a bit. Before, they were using GraphQL image, and now they have only switched. So now it looks very similar to the ORM from the outside. So it feels very much like using the ORM. But it is also that you get the full, let's say, database experience. You have a database inspector. So you can check out all the roles and tables and see how it looks like. And they are handling the migrations and also deployment in the future. So I would say like package together ORM and the full database solution now.

[00:18:33] JM: Alright. So we got Prisma on the backend supporting multiple database modalities. And then you have an opinionated deployment story. Tell me about the deployment story.

[00:18:47] MS: So right now, we are pretty light on the deployment as we were mostly focusing on the application building capabilities. But we also want to be flexible here as much as possible. So what is happening right now with Wasp is, under the hood, we produce the full React and Node.js application. So with that, we actually provide all the frontend files to developer along with Docker container for the backend. So they can just easily put that container, for example, on AWS, or Heroku, or any service, which is supporting it. And they can host their frontend files, so Netlify, or any other server that can support that. So that is right now. So it is pretty much the developers who have to do some work by themselves. But we plan in the close future to add support for all the major providers. So we want to make it easy for developers to just use our CLI to deploy to AWS, or Heroku, or anything in that fashion. We want to make it a more seamless experience so they don't have to go out and manually mangle the files.

[00:19:45] JM: Okay. So put it all together for me. I get opinionated frontend stuff, opinionated backend stuff, and opinionated deployment stuff. What does that give me as a developer? What's my experience using Wasp?

[00:19:59] MS: So yeah, I think it is exactly what you said. It is basically hitting best practices for building web application while still having enough flexibility to use technologies like React and Node.js and write your custom code. So it is very similar to the mission of any web framework, because any web framework has to be opinionated to provide some value. So I would say if you're doing the same, but only be the focus on the current state of the technology, current architecture, and with an extra level of abstraction that can support more technologies, which might come in the future

[00:20:32] JM: Inherent to Wasp is the Wasp compiler. Can you tell me what the Wasp compiler does?

[00:20:37] MS: Exactly. So because Wasp is a language, although a simple one, configuration language DSL, still we do have a compiler. So once you write your Wasp navigation in the Wasp file, basically all that together with your React and Node.js code, everything is analyzed by our compiler, which is doing all the, lexical, semantical and syntactical analysis. And basically on that, we can also see what are the requirements that user developer is hitting. So from that in the future, we're excited to do some optimizations, even compile time learnings and all that kind of stuff. But what is important is that, right now, we are generating the end code in today's module technology. So right now, we are generating the full obligation in React and Node.js along with all the deployment artifacts that are needed. So Wasp compiler, just a part in between, which is from your specification in Wasp and your React, Node.js file generating the end solution in whatever technology of choice it's going to be.

[00:21:35] JM: Tell me, the developer workflow for using this, especially with regard to developer ergonomics. As you said, developer ergonomics is something you like. I'd love to understand kind of the experience. Like what's my sort of REPL kind of thing?

[00:21:49] MS: Yeah, sure, sure. So it's pretty similar to other tools that you might have used or that other developers had used. So everything that has to be done is basically you start with Wasp new and type the project name. And then Wasp generates the initial project in some folder structure for you to get started. And pretty much with that, you can just start adding your code, Wasp code, or JavaScript code. And once you run the Wasp start, it's going to compile all the code and generate the target code, and basically run it in front of you. So it's very similar

just developing your React, Node.js application and running NPM starts in the background. So Wasp is just doing that for you. So once you run the Wasp start, it is listening to all the changes and calling underlying NPM starts and **[inaudible 00:22:35].** So basically you get the same experience of seamless development.

[00:22:39] JM: Is there a way for me to use this on an existing project? Or is this purely I have to do Greenfield?

[00:22:48] MS: So it's pretty much similar as any other framework, so which means pretty much Greenfield. So of course, I mean, you can communicate via API to any other service. And also, I think there is possibility to do some inspection with Prisma databases. So basically, you could start with an existing database. But we haven't gone much into that yet. So for now, it's mostly Greenfield projects.

[00:23:09] JM: And like how much boilerplate does it give me? Like kind of what's my out of the box experience and then kind of my hello world experience? And then how do I go from there?

[00:23:21] MS: So of course, it's hard to put an exact number on the lines of code and how to compare it. We love to say that we are going for at least 10X experience, and like 10X less amount of lines of code. And sometimes it's true. Sometimes it isn't yet. But I would say it's ranging from 5X to 10X, depends on the project, and is going to especially be changing in the future. But I will say what is the biggest wager? It is not just in the sense of the start and having to write less code. I would say the biggest wager is having to maintain less code. Because once you have less code that is written and then you have some higher level abstractions. So you don't have to write all the boilerplate. That means that you are having less code to maintain in the future as libraries are going to change. So I would say that is the biggest value of Wasp moving forward.

[00:24:08] JM: And what kind of feedback have you gotten from people?

[00:24:14] MS: So we have been getting various feedback, especially in different stages of the project. Especially in the beginning, once we have been saying that Wasp is a new programming language for building web applications, we realized a lot of people got afraid of a

new programming language for building web apps, because it sounds like something general, like Python or Java. And now it's going to be something that you have to learn completely from scratch. And you won't be able to use anything that you have known before.

So from that feedback, we have learned that we have to focus much more on the interoperability with the recent stack. So once we started saying, “Look, Wasp is just a simple configuration language, which allows you to specify stuff quickly and easily with less code, but you still have the full flexibility of using your favorite technologies like React and Node.js,” people started getting it much better, and they started becoming much more excited about it.

So I would say that was the biggest learning from us and the biggest shift in feedback. Of course, I mean, with every new technology and every new framework, there is always this moving from alpha version and bridging this gap, coming to beta and 1.0. So this is definitely something that we have to go through. And people can often be worried, “Is it flexible enough? Can you really provide this flexibility of Ruby or Rails and similar frameworks?” So I would say, step by step, we are proving that we can. And also with more examples we have and more applications running, it’s going to become more clear how Wasp is actually running and what kind of benefits is it bringing.

[00:25:42] JM: What are the biggest barriers between where you are today and profitability?

[00:25:52] MS: So I would say for that, the first step is developer adoption. So definitely, without developer adoption and developer validation, if there are no people using it, we cannot think about monetization. So this is also, as I mentioned before, this is what we are focusing right now, building a product and solution, which is going to be very economical for developers, and like very much to their taste. And once we have that, and once we have that validation from other developers, I would say this is a very good cornerstone for thinking about monetization.

[00:26:24] JM: I’m kind of curious about what you're seeing in the capital markets. So this was sent to me by – I think it was Tyler at Abstraction, right? That's who connected us?

[00:26:32] MS: Yes, exactly.

[00:26:33] JM: Okay, right. So he invests in developer tools, and he sent me Wasp. I'm definitely not disappointed. I think what you're working on is really cool. But if you bring this to market and you say, "Hey, look, we're building this full stack framework thing." I don't want to say framework anymore. Developer experience platform. I'm doing a marketing for you. Is this enough? Can you say, "Hey, we want to raise a seed?" Can you say, "Hey, we want to raise a series A?" What does the market need to see today?

[00:27:02] MS: Yeah. Actually, I mean, I would say there is maybe like few thoughts around tools like this. On one hand, there is Ruby on Rails that never monetized solution like that. For example, MeteorJS was a great success. And they were also a company that was going to monetize. And today we have Next.js, which is even the more frontend-based framework. And again, they are very successful story in sense of the company building and even monetization. So I would say there's multiple good models that we can point to and learn from. Especially what we love to use as a model is Terraform, because Terraform is also a configuration language, which is doing abstraction. Only they did it on the sense of the infrastructure. And we are actually looking to do the same, but for the applications. So I would say that's the model we are using the most and going after. So this is also how we are seeing, let's say, how we can show it to other people the potential of the market.

[00:27:57] JM: Yeah, you bring up a really good point. Ruby on Rails was never monetized. Imagine what would have happened if Basecamp would have said, "Hey, we're going to make a hosting platform specifically for Rails."

[00:28:08] MS: Yes, exactly. It's a very good question. And I think that there're a lot of theories and conspiracy theories around that. Why that didn't happen or how it could have happened? So I would say nobody can really tell the right reason, the founder and people closely involved. But yeah, it sounds like a very potentially exciting opportunity.

[00:28:26] JM: Can you tell some of those conspiracy theories? You can discredit them? Or discount them by saying this is a conspiracy theory. But I actually never even thought about that.

[00:28:35] MS: Yeah, I'm just kidding. I mean, I don't know maybe like some super shady conspiracy theories. I would say it's mostly about people saying that it was a byproduct of Basecamp. And people were mostly focusing on Basecamp. So maybe it wasn't their top of the mind to monetize. Also, on the other hand, when Ruby on Rails came out and when things were a bit more focused on server side, I would say there was less of moving parts. So maybe it seemed like there was a less space to add value and potentially monetize. So I would say those are the main thoughts that I kind of encountered when thinking about it and also investigating about it.

[00:29:09] JM: Alright. And so when you look at Vercel business, Vercel just have a series C. I would argue much of Vercel's value is connected to Next.js. Obviously, they have a very comprehensive platform with a ton of interesting roadmap ahead of them. But the Next.js tie in is very powerful, especially when you consider these sort of high-level template-like things that they're introducing on top of it. When you look at Next.js plus Vercel as a platform company, is that similar to what you want to build? Does it mirror that in a way?

[00:29:43] MS: Yes. I mean, I would say in some sense, yes, because obviously they're targeting developers. And they're streamlining the developmental of static websites and, as you mentioned, provide all the tooling for that. So we would love to add value in the similar way, only focusing on the full stack of applications. So I would say that say in that sense, definitely yes, it's a very good role model for us to go after.

[00:30:05] JM: And when you look at the market – So you see Vercel. You see Netlify. I think those are the two businesses that are closest to what you're trying to do. There're a few other things. There are a few other hosting platforms that I've seen that are really interesting. I invested in Render. I really like Render. Where's the whitespace? What is not being covered by these companies?

[00:30:27] MS: Yeah, I would say there's also a difference between hosting and all the tooling that goes around deployment. So I would say, though, for ourselves, we definitely want to make it easy for people to deploy their application to whatever provider of hosting they want. Is it Render? Or Netlify? Or some combination of those? So I would say, definitely, we are not looking to just be our own hosting provider and just support that. I would say we are more

excited about providing all the tooling and like all the tools around for deployment. For example, monitoring, logging, all the best practices around that. So we believe there is value in providing all that, because we know that there is a full stack application while Render doesn't know what you're hosting. So with that information, we can provide a lot of best practices around the deployment part.

[00:31:14] JM: But doesn't Netlify have some kind of script that they run over your code to figure out how to deploy it right?

[00:31:20] MS: I'm not sure exactly what they're doing. So maybe I'm missing some information. But I mean, definitely, I would say they're not covering the full stack of applications or probably cannot do everything from just one single code pass.

[00:31:31] JM: Yeah, too much stuff to do there, I assume. I mean, I'm sort of shortchanging you on purpose, because I want to get to this idea around the features that you're working on. Basically, the idea that your framework at a fairly low-level would be willing to own authentication, pages and routing, blurriness between server and client, which is very modern, smart caching, entity definition, access control policies, NPM dependencies. If I have a full stack thing that helps me with all that stuff and then you have like a big list of other things you could catch, that seems pretty cool. It seems pretty compelling. Could you like sell yourself a little bit further? Why are those kinds of opinionated line items useful?

[00:32:21] MS: Yeah, I mean, we're also are excited about that too. So all that you mentioned I think is also kind of listed on our public roadmap. So I would say with that, let's being the full stack framework means that we have full observability around like all the data models, how they're flowing, the frontend between content. So with that, there is a lot of potential points of adding value. As you mentioned, like from ACL, to blurring the lines between frontend and backend, server-side rendering and all that stuff. So I would say just the fact that we're focusing on the full stack experience, there is a lot of places we could insert ourselves into and provide a lot of helper tools for developers that are using it. So yeah, I would say that is kind of the high-level approach. I'm also happy to go maybe into more details on that if you'd like.

[00:33:08] JM: Yes, go deeper, please.

[00:33:10] MS: Yes, let me just think for a second what might be interesting. I'm also thinking, like the thing is that with all of that, because Wasp is a language. So basically we can provide some compiler. Let's say, for example, compile time and safety for people. So let's say if you specify a page or route that doesn't exist, we can warn you even before you run the code, like, “Look, there is no that page. Like you haven't defined it anywhere.” So that is just one example of compile time safety that we can offer.

On the other hand, again, with owning data models, which are being defined in Prisma. And then, basically, we can help you easily like have all those models from the Prisma models, and database models, to backend models, to frontend models. So I would say just there is already a big value. So if we can make a **[inaudible 00:33:54]** solution for that, it's going to save a lot of boilerplate and also a lot of, let's say, possible security issues. Especially, once we had the ACL access control list, which is going to allow people to say on multiple levels like this property is being protected. It can be only on the backend. On the other hand, like these queries and actions are going to be only for the authorized users or for the users with a specific role. I will say, with all of that, we could streamline a lot of development right now and also make sure, let's say, it is secure and follows all the best practices.

Another thing is maybe testing it, because right now, setting up tests and making sure they runs smoothly, also with end-to-end tests, there is a whole mess around that, and kind of the whole separate workflow. So we would also love to provide the solution for that, which comes out of the box. And even knowing some of the requirements from the language part, we could also provide some initial end-to-end tests for you like just from the beginning. So I would say, with all of that, we could automate a lot of the current development process.

[00:34:59] JM: And how deep can you go? How far can you go? Actually what I should ask first. So, to my mind, this is sort of the defining characteristic of what you're doing, because there's an effectively infinite roadmap of things like this. So for example, in your coming next column, you say, eventually, we want to have auth methods like Google and LinkedIn. So if I have a framework that bakes in Google and LinkedIn auth and I don't have to go to using whatever module, like Firebase or – What would I use? Like Auth0 or something to implement these sorts

of things? Like kind of, arguably, if you compare to implementing this at the framework level – Or sorry. Wait? What do I call you again? Not a framework, but a what?

[00:35:43] MS: No, I think it's okay to go into framework just kind of to distinguish between the approach. But I would say framework, again, it is not a bad word. So feel free to continue using that..

[00:35:52] JM: All right. So if I get like Google auth at the JavaScript framework level, that's really appealing to me.

[00:36:00] MS: Yeah. Again, it's not by default that we are excluding other possible solutions. For example, I think it's very believable that we are going to have in the future support for using Auth0 if you want to use it, because it's a great tool. It is just that you create easily with Wasp. And again, Wasp is going to do majority of the bullet point connection with that. So I'm not sure how it's going to go exactly in the future. But I will say we are not excluding that. Actually, we are counting a lot on using the existing solutions and integrating with them.

[00:36:30] JM: And just to jump out at another thing that is coming soon or coming next in your roadmap, server-side rendering. So server-side rendering to me feels like basically a bottomless problem. Because if you think about kind of ideal use case, ideally, you have the server-side rendering all of your pages, right? I mean, like in an ideal world, that's not really possible. But that's what you would ideally have, right?

[00:36:58] MS: Yes, exactly, exactly. Ideally, you're in frontend-side rendering, because it's kind of taking more time. But yeah, with Next.js and the other solutions, let's say, it's now very popular again to go into server-side rendering after people have experienced the problems with single page applications. And I would say it's especially relevant for sites with a lot of content. So basically, you don't want to wait all the time for that content to come and to render, especially for sites which are open to anybody to just come and read all that content. So I would say, for us, on one hand, it was not super critical in this stage when we're focusing on just single page applications, which are typical crud applications. But also moving forward, we think it's going to be important for being able to have nice and good performance in applications. So we definitely want to make it easy and smooth for people to use that.

[00:37:50] JM: And if you are able to bake in something like Google auth at the framework level, presumably, that kind of work is going to further your server-side rendering efforts, right? Like if you can make more assumptions about what's going on in the infra, you're going to be better at rendering on the server?

[00:38:11] MS: Well, yes, I think so. Because the know we more in advance, the know we can prepare on the server and just send it to the client.

[00:38:19] JM: So the language of your DSL, because this is a DSL. Actually, we should double down on that for people who don't really understand what we're talking about. You have a .wasp file format. Why do you have your own file format? Explain what a DSL is more broadly, and why it applies here. And like what your DSL does?

[00:38:45] MS: Sure, sure. I'm happy. So just to start with the formal definition of DSL, or domain specific language. So it means as suppose – So we have general programming languages on one side, which is your typical Java, Python or JavaScript, which are basically Turing complete languages, which means you can write any computation in them. And then on the other side, you have domain-specific languages, which means they are not as powerful as general programming languages. So you cannot write any computation in it. Maybe you don't even hear for loops or if statements, but they are super well-designed for a specific use case.

For example, one of the most famous DSLs that we are using, a lot of us are using every day, are SQL for databases. And on the other hand, HTML for writing web pages. So I will say, with that in mind, they're super well-optimized for their job and they're making our lives a lot easier. It's more economical to write, it's less code, and to generate everything with some general programming language. So that is the main purpose, to be very specialized for one purpose and make it easier to write and maintain.

So the same thing is actually with Wasp. And we like to compare with Terraform, because we are similar in the sense of the language and the language structure. So like it is a configuration language. Very simple in its nature, very declarative. So basically, you can just say I have a page. It's using this via component. I have a route. It's pointing to this page on this route. And for

example, I'm using authentication. And I want to be able to use Google or LinkedIn, and email and password. So that is just to give you a feel of how Wasp DSL looks like.

We can even say, right now, it is like a nice JSON. And you could probably have one-to-one connection with JSON. But we're just hearing this format, because it is easier to write for developers and also like more human readable. And just pure JSON, which easily and quickly becomes very nested.

[00:40:38] JM: What's it been like developing a DSL? Like that's something I've always wanted to do, but had been scared to. I took a programming languages class in college. That kind of scared me, because I didn't do very well in it.

[00:40:51] MS: Yeah, yeah. No, you're right. I mean, it's pretty exciting. And yeah, we also had a programming languages course in our university, which was actually quite good. So I think we got a lot of good foundations for that. And as I mentioned, Martin, and I were always kind of focused a lot on core computer science problems. So we got a good foundation from there.

So yeah, with DSL, in the essence, it's similar as building any programming language. So you have to have a parser. You have to define a grammar. You can parse the ASD, which is like the syntax tree. Basically, like the structure of your program. And in end, generate the end code in some lower language. So in that sense, Wasp is similar. But of course, since it is not a full programming language, it was much easier for us to get started and to build the whole thing.

[00:41:40] JM: Do you have to do the whole abstract syntax tree and stuff?

[00:41:44] MS: I mean, it kind of appears naturally. So it is not something that complex and mythical. For us, we are a compile language, which means basically we compile everything, and then made decisions before we generate the final code. So I will say we are not like typical interpreter, which is going line by line and doing stuff. For us, it is actually very important to go through everything, because this is pretty much – Wasp language is a requirement of your applications. So once we can go through that, we can analyze. And what do we have? What files do we have? What kind of specify that you want? And we can take a look at everything from the outside and then decide what kind of application we need to generate. So for us, it is

important to have that step. And as I said, it is not actually that complicated when you are implementing it and using it.

[00:42:31] JM: What's the status of the company today? What's the hardest thing you're working on?

[00:42:38] MS: Yeah, I would say there're multiple hardest things that we had in front of us. I would say, especially before, it was getting the language on the ground and making sure that our initial abstractions make sense. Also, we're using Haskell to build our compiler. So we have been using it before, but it was never like on the full scale production project. So we also need to take some time to learn it, learn the bases, figure out how to set up the project. But once we have done that, actually, we are now pretty quick. And we are utilizing all the features of high-level functional language. So it seems to be a really good fit for writing a compiler. So we are very happy that we chose it for this task.

On the other hand, of course, it's mostly about communicating with developers, setting up the right strategy for that, setting up all the right channels. So we are putting a lot of focus on being very open source friendly. So we are making sure to put all issues on GitHub. Foster discussion on our Discord community. So I would say we just want to make sure that we are making these things right, and that we are being open and inviting to developers and giving them all the tools they need to engage with us and start contributing to the project.

[00:43:48] JM: When a developer tries out Wasp and they say they don't like it, or you see that they churn out, do you have an understanding of why they're doing that?

[00:43:58] MS: Yeah, I mean, were' trying to get it as much as we can. So it's very important to get the feedback. Actually, what we have experienced is that people are much more skeptical of Wasp before they try it. Especially at the beginning, as I mentioned, people were afraid of this approach with a new programming language, and they thought it couldn't be flexible enough, and it's going to be too much to learn. So I would say most of the people who didn't like it were the ones who haven't tried it beforehand. So once they tried it, a lot of people actually gave us positive feedback and said it was easier than they expected.

[00:44:31] JM: Let's zoom out. I want to just ask your opinion on some stuff going on in developer tooling. We talked a little bit about the hosting platforms, Render, Vercel, Netlify. So have you looked at Replit? Do you know Replit?

[00:44:50] MS: Yes. I do a bit. I haven't used it extensively. Maybe just tried it once, but I know the general idea.

[00:44:55] JM: Okay. Okay. Well, Replit, the thesis of unifying the IDE and the deployment platform. To me, that seems unparalleled. What do you think?

[00:45:08] MS: So you mean it's a very good idea when you say unparalleled?

[00:45:11] JM: Well, I just mean, if you're thinking about from like – So I love all these companies. I think they're all going to do great. I think they all have their moats. But when you think about who has the most differentiated mode as a next generation hosting platform, I mean, if you get an IDE closely integrated with a hosting platform, that seems very powerful. I guess GitHub is sort of doing that also. What happens when we start to have the IDE integrated with the deployment platform?

[00:45:38] MS: It's very interesting. I would say, especially like – There is now a lot of tools like Replit, code sandboxes which allow you to quickly get started and give you some initial environment. And you can even, as you said, code anything from the browser. I mean, actually, I have to admit, I haven't used it a lot. I mostly use the JSFiddle for trying out some JavaScript quirks and stuff like that. Maybe I'm a bit more old school developer. And I love to use my Vim and my terminal locally. So I will say, for me, for production projects, I still prefer to have my local setup.

And from our feeling kind of and in kind of our developer circles, we know that they love to have that control for themselves, especially when working on the production-level projects. But I also do believe that in the future, things are going to become much more integrated together, as you said. Now it is just the question of what is going to be that path, which is what is going to be the intermediate step? Which is kind of changing us from more locally-based environment to

altogether integrated environment, which I would have to think a bit more about that to give maybe a smart answer on that.

[00:46:46] JM: Any other developer tooling systems that have really stuck out to you recently? Sorry to put you on the spot.

[00:46:53] MS: No. No worries. I will say though, there is a lot of stuff. And especially me and Martin, we are big fans of open source. And we have always loved a lot to like sharpen our developer tools. As I said, I'm a big Vim user. Martin is an Emacs user, setting together our kind of workflow and all the stuff that we're using. So maybe one big thing for us it was language server protocol that is right now being popularized and used by all the editors. So that is also no one attempt and successful attempt in streamlining the solution in the sense of editors. Because before, every editor has their own completion for different languages and different solutions. And right now, with this approach of the language server protocol, which means that you're having one logic for providing autocorrelation for a certain language, for example. And I think that is brilliant. Because with that, it is not that every editor now has to implement their own solution. They could just plug into this existing LSP, language server protocol, and get all the goodies from that. So I would say, I think that is great. I love it. And this is, in some way, also how we are thinking about Wasp. I'm going to say, Wasp being that protocol for defining abstractions in the sense of application. And then again, different tools can plug in with that.

[00:48:13] JM: Do you know what – What is GitHub doing right now? Like what is GitHub going to be? What is it offering us in terms of developer abstractions? Like I saw this AI coding thingy yesterday that launched, but I was traveling, so I couldn't really go into it in detail. That looks kind of important.

[00:48:30] MS: Yeah, definitely. I mean, GitHub, as I said, like they're also tried to integrate more and more, which is completely logical with what they are doing. We are also using GitHub actions for our deployment and building stuff. So I think it is very useful. And I think they are going to continue this direction. To me, it's also logical that they offer some sense of VE that is easily available, and with all the prepackaged solutions and environments easily to use. Again, something in line with Replit.. And they are definitely in good position to do so. So I will say, with

us, Wasp is maybe like another language, another solution, which might be the thing that GitHub is supporting. So I would say we are just another language being hosted on GitHub.

[00:49:12] JM: Alright. Well, we're nearing the end of our time. If you were to give a summary, for our conversation, why people should care about Wasp and what you're building, can you summarize everything?

[00:49:25] MS: Yes, sure. No, I'm going to do my best to do it. But yeah, I will say with Wasp, what we are trying to do, we are trying to – I mean, we are doing the abstracting, developing web applications. And we want to make it possible for developers make web apps with less code and with less boilerplate, and also still letting them to use their favorite technologies like React and Node.js, which is what we are supporting right now. So let's say our big vision for the future is to be at the abstraction layer for building web apps and support whichever technology is the latest state of the art stack.

[00:49:59] JM: All right. Well, It's been a real pleasure talking to you. Do you have anything else to add?

[00:50:02] MS: No. I think everything good. And also, thank you very much for inviting me. I enjoyed the conversation. And you had very thoughtful questions.

[00:50:09] JM: Okay. Look, it's been real pleasure talking. And I'll be following it closely. I kind of wish I could use it on my current application. Maybe I can. The extra compilation step kind of worries me. It's a little unfamiliar to me. Is that okay?

[00:50:25] MS: Yeah, definitely. I would say what you should worry more is that we are still in alpha right now. But yeah, compilation is very straightforward. And like any tool right now, when you run NPM install, a lot of stuff is just kept running in the background. So I would say we are not that much different than other tools.

[00:50:41] JM: Okay. All right. Well, hopefully we can check in six months or a year down the line and the product is completely different.

[00:50:48] MS: Yeah. Yeah, definitely. Happy to stay in touch.

[END]