EPISODE 1149

[INTRODUCTION]

[00:00:00] JM: Firebase is a well-known platform that makes it easy to build real time applications quickly and easily. Firebase was acquired by Google and has been turned into a large platform that runs on top of Google Cloud. Firebase is closed source, which leads to a different ecosystem than open source platforms. Supabase is an open sourced alternative to Firbase built on Postgres and Elixir.

Paul Cobblestone is the founder of Supabase and he joins the show to talk through what he is building. We talk about the architecture of Supabase and how it compares to Firebase.

[INTERVIEW]

[00:00:39] JM: Paul, welcome to the show.

[00:00:40] PC: Hi, Jeffrey. Nice to be here.

[00:00:43] JM: It's great to have you. You work on Supabase, which is an open source Firebase alternative. Some people might not even know what Firebase is. Could you explain what Firebase is?

[00:00:53] PC: Yeah, sure. Sure. Actually, Firebase is quite a good product. Well, it's a very good product. Basically it gives you everything that you need to get started on a project. So when you sign up, you can get a database, sort of NoSQL database. Then you get authentication. You get analytics. Basically, all the things that you'd need to start building a product. Firebase gets you out of the box.

Now Firebase started actually in YC about 8 years ago, and it was acquired by Google. Since I think around 5 years ago, Google has been building upon it, but it's fully proprietary, except for the client-side code. So we're building an open source alternative. We've made some technological decisions, which are slightly different from Firebase. But the feature set that we're targeting is roughly the same, and the experience that you have as a developer where you can

sort of get up and running in just a few minutes, that aim for Supabase.

[00:01:48] JM: Why does it make sense to have an open source Firebase alternative?

[00:01:52] PC: Yeah, most of all because what happens with Firebase, at least if you look online, you search, there are a couple of problem with Firebase. First of all, it's locked-in to Google, which isn't necessarily a bad thing if you want to stick with them. But the other thing is because of their choices, it doesn't scale very well. So you'll see that this means several different things. As you get to a really enterprise-grade, then actually the technology underneath it doesn't necessarily scale. So the number of reads and writes per second might not be enough.

But also, maybe they're billing doesn't scale for some people. You might have just, say, a podcast with 30,000 people and lots of people signing up and suddenly your bill goes through the roof because they charge on usage. So an open source alternative gives a few things. One, it gives you full control of their underlying technology. And two, it gives you also an ability to migrate away if and when you want to. We hope that you wouldn't need to, because we are choosing technology, which can actually scale up to enterprise-grade. But yeah, you always have that option if you use Supabase. And three, it's just great to contribute back to the open source community.

[00:03:10] JM: So there's so much engineering that has gone into Firebase. What are you focused on starting with?

[00:03:15] PC: Yup, great question. The database. So database is the number one thing that you kind of have to start with. It's focused on where you store all the data for your business. It's probably going to be one of the things that you have to think of first. It's also one where there's a lot of need for this Firebase-like experience on open source tools.

So what happened at the start of this year, we actually started in January. We spent a lot of time chatting to engineers, and most of them would say, "Yeah, I really like Postgres," the database. And then we'd say, "Well, what are you using?" And they would say, "Well, we're using Firebase and we're doing that just because it was the easiest to get started."

So we started with the database. We started with Postgres, which we always intended to as open source as they come, and it's backed by sort of 40 years of great technical engineering. And it can scale definitely to enterprise. So our goal was just to make Postgres as easy to use as Firebase. So at the very start, we were just measuring how quickly you would sign up to Firebase and then spin up a database. And then our goal was to make that exact experience similar with Postgres so that you could sign up, spin up the database directly on our dashboard and then query it directly from the dashboard rather than having to install multiple tools and knowing what Postgres is. You can just literally start a project. And without every knowing the underlying technology, you are able to start using it.

[00:04:48] JM: So if I recall, Firebase was originally Mongo under the hood. Correct me if I'm wrong, but you use Postgres under the hood. Is that correct?

[00:04:56] PC: Yeah, that's right. They started with Mongo for their real-time DB, and now they've got another one called Cloud Store, which I believe – I tell it to James. I think he said it's a proprietary, one of the Google databases that they're migrating over to. So we use Postgres, correct. Of course, that means things are done slightly differently, because Mongo is a NoSQL store and ours obviously is a relational database, which is a little bit more complex for newbies.

So what we've done is actually give – If you've ever used Airtable, we give a similar sort of experience where you can sign up. You can start setting up your tables directly from the dashboard. You can just add tables like you would in Excel spreadsheet. You can add columns just like an Excel spreadsheet. We kind of guide non-engineers towards setting up and using the database.

But at the same time, if you are a full engineer, we give you literally a Postgres label access. One full server, not a schema within a database. We give you a full database. And if you want, you can just log in to your database. You can set it up using migrations. You can set it up on the dashboards. You can query it. So we're literally catering for all audiences.

[00:06:07] JM: Take me through the basics of a read and a write to Supabase.

[00:06:12] PC: Oh, yeah. Okay. That's a great question. So what happens with Supabase – It's probably easier to start by explaining the stack from bottom-up, and happy to go deeper on each one of these. But basically, we start with the Postgres database at the very bottom level. So there's a relational database. Then from there, we have an open source tool called Postgrest, with a "T". And this introspects your database schema and it auto-generates an API. Not unlike GraphQL. So it basically just provide a very thin layer on top of Postgres to give sort of HTTP functionality.

Then on top of that, we have API gateway called Com. And that's another open source tool. And then finally on your client side, you can either install a JavaScript library, which we are putting out ourselves officially. Or a lot of client libraries have been developed by the community right now. So there's a Python 1 I think still under development. Someone's developing a Dart 1, Rust 1 sort of being kicked off. But let's say you use the JavaScript. So you install – If you've got some sort of Jamstack, for example, maybe you produce something on NextJS or you put something on Netlify or even Webflow. Anything where you're going to store a JavaScript client, you can then say something like from table select all these columns including all my relationships within those columns, .filter and then it all pass back all that data. And you can sort of think of it like a very, very thin layer on top of a SQL query, but it looks more like a JavaScript library.

[00:07:57] JM: What is difficult about building a real-time database?

[00:08:01] PC: This is actually kind of the origin story of Supabase. So we were using Firebase in my previous startup. So I had developed kind of a WhatsApp-like chat application. And then we hit some scaling problems with Firebase. Ours was a particular – It was kind of my fault. It's hidden away in the docks. They've got this limitation of one query per document per second. So I hadn't structured the data correctly, and I didn't know that. So I had to reengineer how I've done the data model on Firebase to overcome those limitations. Or I had to migrate away. So we're already using Postgres for most of the functionality. So what we did was I just migrated all into Postgres.

Now Postgres doesn't have built-in real-time functionality. So what I did, I started very naïvely. Basically, I would just listen to any updates. You can sort of put a listen functionality or triggers

on insets and updates to a table. And then I'd do this thing, Postgres has this thing called notify, which will send out. You can just listen to notify signals. So what I was doing is just listening on insert. I would send a notify out. Now, this had a couple of problems that I didn't realize until much later. The notify in Postgres has a hot limit of 8000 bytes. So I was sending out JSON documents, which would grow much larger than 8000 bytes.

So I ended up reengineering this. And what we did, I found a small library that it kind of kicked it off, and what we did was we created an Elixir server, and that Elixir server kind of acts like a sort of Postgres databases. It listens to the replication stream of any Postgres database. And that comes in binary format. Then it convicts the binary stream, the write ahead log replication stream. Converts that into JSON, and then it blasts it out over the WebSockets.

So the Elixir server is built with Phoenix, if anyone is an Elixir fan. So that one can scale very well. You can connect. I think there's the dumb benchmarks of 2 million WebSockets per server. And then we just have a single listener to the database. So this became sort of the perfect solution for our use case where we just have one listener to the database rather than lots of different connections in. And then everyone connecting to the Elixir server, which is hugely scalable. And so what happened was I open sourced that actually in September last year and had started gaining traction. And that's the reason why in January this year we actually started to the base.

[00:10:42] JM: So if I understand correctly, you've got basically Postgres database with Erlang on top of it, or Elixir, which is the Erlang dialect. And there's just a single connection between that Phoenix server and Postgres, and then you accept multiple connections so different clients can subscribe to the Elixir server, but you only have a single write path to the actual database.

[00:11:10] PC: Yes. So the write path, this is listen only. You noted, by the way, that is exactly how it works. So except for the write. The write path goes through Postgrest, the tool that I mentioned earlier. So this one is just a listener. It listens. It is actually a change data capture tool. So it will listen and it will send it off to multiple system. So one of those systems can obviously WebSocket subscribers. But we're actually building in that it will listen to your database and it can even send it off to, say, webhook, Kafka, SQS, any queuing system that you want as well.

[00:11:46] JM: Very cool. So beyond the Elixir server that sits on top of Postgress, what have you had to build to make the real-time database work?

[00:11:56] PC: Everything built into the Elixir server, and then we just hook into Postgres' replication functionality. So really, I got to say, we're moving faster this year. So we've been doing six months of building of our hosted platform, and we're quite functional really. You can start using us. But, really, it's because we're standing on the shoulders of giants. I mean, Postgress, and the amount of functionality it gives you out of the box is sort of cheating for starting a business on top of it. Of course, it's purely open source. So Postgres does most of the heavy lifting here. Even the Elixir server is just – Literally, it's quite dumb and that it just converts a binary stream. It listens and converts it and then blasts it out. So there's not much more than that on the Elixir server right now.

[00:12:40] JM: So where is your work focused today?

[00:12:43] PC: Yeah. So we've got this hosted platform. Basically the experience for you as a developer, it might be that you'd come on to our website. You sign up using GitHub, and then you start an organization which you can invite your team members to. And then you start a project. And when you start that project, we actually spin up all the backend infrastructure for you. So individual databases, all the pieces that I mentioned earlier, we stitch it together. We're bringing out replications so that you can start a database in multiple regions around the world so that you cut down on latency for your users.

The dashboard itself, which has multiple parts to it. Sort of this table view that I mentioned to you. And then finally, we spent most of the past, I think around now three months, building an auth. So that was quite a funny story. What happened was we got into the previous YC batch. It's just finished up actually. And the day that we got in, we sort of updated our website just to say backed by Y Combinator. And someone put it on Hacker News, and we had this quite successful launch, sort of an accidental launch. It wasn't intended. We just wanted more, a few more sign-ups, and we ended up getting thousands more sign-ups. But we're very unstable at that stage.

But the good news is that from their Hacker News launch, we could just tell the things that everyone wanted, and everyone was asking for auth, because at that stage we just didn't have a good auth solution. So we spent all about time in YC. Instead of focusing on growth, we just focused on building auth, and we release that about a month ago. And yeah, we're still tying up the loose ends there.

[00:14:28] JM: Naïve question. Why is anybody using Supabase? Why wouldn't they just use Firebase? Firebase is so much more richly featured today. Anybody who's starting up a new project, why wouldn't they just jump on Firebase? It's got a bajillion more features. I mean, I get the long term vision that if you use the open source Firebase alternative, then you can kind of like have a better scalability story that's cheaper. Maybe you can like spin up your own thing. Maybe you fork it and do some kind of thing under the hood. But for most people getting started, they just want to use Firebase. Why would they use anything else?

[00:15:10] PC: Yeah, it's a fair question, especially because we're still in Alpha. But there are several reasons. I mean, first of all, the very basic question is why are they using Supabase? One of the reasons quite honestly is because engineers like to try new things, and we are a new thing, and they just want to try us out. So we do get quite a lot of these type of people come and kicking the tires, like what they see, and we'll just continue.

The next one is Postgres. Honestly, as I said before, Postgres is kind of cheating in terms of functionality. So if you just wanted a free database, a free Postgres database, you could just come in to Supabase, sign up for that, and then you would get this database free of charge. But then we've got all this additional functionality on top of it. So as you start using it, you'll realize, "Oh, actually, it's kind of handy."

So as an example, when you start setting up your tables, the auto-generated API also in your dashboard has auto-generated documentation. So you can literally just copy and paste snippets of code into your product and get up and running very fast. But that doesn't really answer – Well, I guess it kind of does answer compared to Firebase. Firebase has some documentation. It's not fully customized, whereas ours is definitely trending down that path.

I think in the long run, if we're to do a Firebase versus Supabase comparison, Firebase definitely has more features and it's more stable. You'd choose us because we are choosing different technologies, because we're open source and because there is no lock-in. So I think in the long run when we wipe out the feature parity argument, then it just becomes a very obvious choice.

[00:16:50] JM: Interesting. That's a very, very long roadmap though, right? I mean, you think about Firebase. They've got like hundreds of people working on that at Google, right?

[00:17:00] PC: Yeah. That's true. This is the story of many startups going up against someone like any other established player. So we do have a long way to go, and we also sort of going to have to compete with Firebase on Free Tier, which is extremely hard for an open source tool. But at the same time, we are using open source tools. So one of our core, core philosophies is don't reinvent. We try to support existing communities wherever we can. So this is true of almost every part of our technology, the database is Postgress, the API is a Postgrest. The auth, we're using Netlify's Go True. And yeah, contributing back as much as we can. We'll be doing some other things around some very established open source communities as well to make sure that we contributing to them. But in return, we essentially get this very rapid development roadmap and we don't just have 8 engineers in our team. We essentially got thousands of engineers across the whole world contributing not just directly to our product, but to the products that we are using ourselves.

[00:18:11] JM: What kinds of advantages are there to being built on Postgres? Are there people who want to use the Postgres APIs under the hood?

[00:18:19] PC: Yeah, I can just talk to the future set or Postgres, which obviously is a relational database for some people. It seems hard. Really, it's not that hard to get your head around. So once you get over that, then you get a lot of advantages. So Postgres is obviously extremely scalable if you're an enterprise and you migrate off Oracle. You're probably going to Postgres. So it's got this hugely scalable path that you basically will never need to worry about it. We've got some people in our system just using it. Well, really, abusing it in the amount that they store inside it and how they use it. And it just handles it. So there's that feature-set-wise. It's pretty amazing.

So it's got JSON-B documents if you want to actually use it like a NoSQL Store. It's got row-level security, which we actually make extremely easy to use. So the security model and down to the row level is top-notch. It's obviously got things that are helpful for you, like generated columns. So you can actually extract out bits of data from different columns or do calculations on multiple columns and install them on disk so it's very fast to read and write, materialized views. Yeah, you really have just got 40 years of some of the best database minds putting engineering and still releasing. I think they released a new sort of major version at the moment, it seems like every six months. And each one is just amazing in terms of the features that are coming out.

[00:19:50] JM: Is the Postgres database replicated under the hood?

[00:19:54] PC: So the one that we give to all our customers when you sign up has replication enabled, and we need that for our real-time listener. So what we will do, who actually end up choosing a location to spin up the database. So let's say you've got customers in New York. You choose to spin up your first database in New York. That's the base of your project. And then what we're going to bring out is let's say you've got customers all around the world. You've got them in London and you've got them, say, in Singapore here as well. You choose to replicate your database across to these zones as well. And we'll handle all of that for you and make it super easy. And then that obviously cuts down on latency for your customers in each geography. So it's enabled, and then soon, probably in the next six months, we'll bring out this very simple replication functionality.

[00:20:47] JM: You mentioned earlier that auth was one of the things that you built that took some significant time. Can you tell me more about building auth?

[00:20:54] PC: Yeah. There are a lot of sort of POC. So as I said at the start, we really wanted to support existing communities, so rather than just build from scratch. So one of the things we had to do was find an authentication. We tried out a few – So shout out to Keycloak, which is an amazing tool for authentication. And also a new one, Kratos, which is also looking very promising, just a bit early.

We ended up using this tool from Netlify called Go True. It's just as simple Go server that handles sort of the JWT functionality where you can sort of sign up. It handles a few OAuth providers and just lets you into the database. And that was good for us, because written in Go, it's quite light. We can actually spin it up for each database.

And then what happens is, all right, once you are into the database, basically we promise sort of this no middleware if you don't want it. So that obviously means that you need to put your rows somewhere, your authorization row somewhere. Now Postgres has row level security where you can literally specify, with this logged-in user, with this user ID, can only insert into this table if their ID matches user ID on this table. And this is great for a number of reasons. One, all your rows sit down in the bottom of your database, or your tech stack. So you can – Essentially, if you're using even say a bash curl command, still the authentication is happening even on the database level. It also cuts down all your middleware. First of all, you wouldn't need middleware if don't want. You can just use, say, a Jamstack type approach or an HTML page and us. But if you do use middleware, say, like serverless functions in between. You don't actually need to then specify all the auth there. You don't need to put all the filters if this user only give me messages where this user exists. You can just say select all. And when the user's logged in, it'll only give you the messages that belong to that user. So it actually drastically reduces your code.

You don't have to use the row level security. You can still put it into the middleware as you would in the past. But this is how we decided to engineer it. So basically there's two components. Once again, to recap, there's authentication using Netlify's Go True server. And then there's authorization using Postgres row level security.

[00:23:23] JM: What has been the most difficult engineering challenge of building Supabase so far?

[00:23:28] PC: Yeah, engineering-wise, the auth was definitely one of the trickiest parts. One, just figuring out how to stitch it all together. But I would – It's funny, because as an engineer, it'd cost you basically fighting fires every day. It's hard to pick one. They'll just sort of blur into one. But I can give you a couple of funny stories. So, one the authorization authentication piece was definitely a challenge. It was a lot of schlep getting through it and making sure that it works and works consistently for everyone coming up with a solution that is very generalizable.

Another one of course would be – I mentioned at the star that we had a sort of accidental launch. When we first started in alpha, we were using DigitalOcean. They were the first to give us some credits. And when we had this launch, I think overnight, we ended up with something like 800 databases. And then of course what happened was we couldn't get any more credits for DigitalOcean. We had only 10,000. So we're about to start paying for a lot of infrastructure. And we're just growing quite rapidly. So they told us this, that they couldn't extend the credit. So we had to migrate 800 databases basically in one week over to AWS where we did have 100,000 credits through their startup program. So that was definitely a tricky one.

Another quite interesting one, because we give Postgres level access, our idea is to give full control to everyone. One of the things we naïvely did was said, "All right. You want to set your own password," but we didn't give people restrictions on their password, or at least not tight enough. So people were sitting just very relaxed passwords on their databases, and then they are essentially just getting pummeled and hacked into.

So what was happening was people were installing crypto miners on the postgres database. You can actually use kind of scripts to give – Once you got Postgres level access to install a crypto miner directly on to the Postgres server, and that was happening just not by the users themselves just because they have not secured their database enough. So a lot of what we do now is just focused on security. Making sure your data is safe. Making sure it's not exposes the world, making decisions for our customers where they might not be making the best decisions themselves. We try to make sure that we safeguard everyone and their databases.

[00:26:00] JM: You mentioned a massive 800 database migration. How do you do that?

[00:26:05] PC: Nothing to over the top. It's just – Once again, engineering time, a little bit of planning. Obviously we're a bit under the gun because we're starting to get build for it. So all we did was basically light migration of the databases, and middleware is on stateless. So it's just a matter of repointing it towards the new database. Any databases which were not active is just a backup and restore. And any ones which were active, we were working closely. We would notify the customers that there is going to be a potential downtime that we just did a quick switch

basically. Then it's just a repointing of the URLs towards middleware, and our middleware towards the new databases.

So the likely thing is we are in alpha and we were definitely very alpha at that stage. We could work just closely, but this was sort of this expectation of not a hundred percent uptime or 99.999 of uptime. So yeah, it just all hands on deck to make sure that it went smoothly.

[00:27:08] JM: Is everything else difficult about getting Supabase to work on the cloud?

[00:27:15] PC: The hardest part of our job is just, one, finding the tool that will scale the enterprise, and then making it work in a generalizable way. So our engineering challenge is not just solving a need for our business. It's solving a need for all the thousands of businesses that are signing up. Often, these challenges are very different from just an individual use case. The row level security for example on Postgres is a good example of that. That's one where the engineers had thought obviously how can we come up with a general rule that works across many businesses?

The way that we structure our code and we structure, the way that we manipulate you database before starting it up, another core philosophy, for example, is that we try to keep your databases clean as possible. So we don't do anything to it, but we had to inject an auth schema into your database now when you sign on so that you can immediately start getting sign-ups. So a lot of the time, it's just throwing away solutions that we come up with, because they are not generalizable to thousands of businesses and trying to come up with these solutions that are. And that's where a lot of our time is spent.

[00:28:28] JM: So you got the real-time database pretty much locked down. What's the next phase of Firebase feature parity?

[00:28:35] PC: Yeah. So at the moment, we're just focused on moving from alpha to beta. So we plan over the next one month to stabilize the product. We do have – After that, we let the community guide us in terms of features. We do have some ideas of what could be next. Some of them are turning our real-time engine into a workflow engine. So you can actually send it off and do simple workforces.

So for example, kind of like a Zapier-like system directly in our dashboard using the real-time server. This one is slightly different from Firebase. So I think over the long term we might – Well, we will diverge from a lot of the things that Firebase are offering. A good example of that is the hosting. So Firebase obviously do hosting, which we have not really any intention to do. We're planning to partner with the Jamstack platforms and the ones who already do it far better that we would ourselves. So Netlify vs. – I don't know if you're using Webflow or any other hosting provider for static site hosting. So firebase provide this, but we will probably never do this.

Another one that is consistently asked for is storage. It is very hard to find sort of an open source storage. Concepts doesn't really exist, because you need to store huge amounts of data. So we may have to come up with a solution for that for our users, because most people need it, or at least find a partner in that regard. Then another one that I think would be quite nice that we're looking into. So, I mentioned that we employ one of the maintainers of Postgrest, and we actually install Post, just geographical information system and inside our databases. So if you use any geographical data, then you can sort of use this extension on Postgres to do a lot of mapping or do a lot of geographical spatial indexing, these sort of things.

Maintainer now is implementing a RESTful functionality on top of this. And as a result of this, we will look at supporting the OpenStreetMaps community to sort of plug tiles into your geographical system. So we plan to make it extremely easy to get up and running with mapping, which is once again quite different from what Firebase is doing. They obviously have Google Maps, which is not a Firebase solution, but we will try to support OpenStreetMaps.

[00:30:58] JM: Interesting. When you said storage, what did you mean?

[00:31:02] PC: Yeah, storage like let's say you got some photos and you want to store them. So avatars for users are a classic one, where you've got someone who signs up and now they want to upload the profile photo. So this one, of course you could store in your database. It's actually possible. But databases are usually more expensive storage. You'd want to be storing it in a very machine storage system, maybe like S3 or Backblaze or something, something very cheap. Yeah, because there's no open source storage system or none that I know of, then we'd

have to try to find a partner or we would host the storage ourselves maybe on our own S3, and then pass along the cost of that to our customs.

[00:31:49] JM: And then would you expose some universal bucket storage API?

[00:31:54] PC: Yeah, exactly. So each time we implement a service, we attach it to one about client libraries. So for example, you can select from your tables, but then you can do, say, supabase.auth.login for different users. So we would do something like supabase.storage.upload, or download, or whatever you want to do. So sort of an equivalent to what Firebase has on their client library.

[00:32:18] JM: What you think of cloud functions? I know in Firebase they have Firebase functions. Do you see that as relevant to your roadmap?

[00:32:27] PC: Yes, we definitely need to implement functions. So there're two things. So at the moment we're doing a POC on this. So we can actually trigger. You can call over your RESTful API or your database functions. So store procedures. You can pass parameters, and actually you can write database functions with NodeJS, Java, Python, or SQL.

At the moment though, so we're installing PO V8. This one, we're going to test whether it's going to be scalable or not. PO V8 is actually no good for writing functions, because it runs in – You can't actually use installed modules. We can, but you have to install them on the server and then you have to run them.

So what we're doing is we're checking whether we can actually run. If you know what Deno is a new sort of note engine by the creator Node actually? We're seeing whether we can create a Postgres extension using Deno. This is very new. So don't hold me to it. And then we are going to see if we can use Deno to actually run – It can run Typescript. It can run packages directly from things like Skypack or JSPM, so from sort of CDNs directly. And then you can be running your sort of Typescript code directly inside your database functions. So we're going to test that. If that doesn't work, then we'll definitely implement some sort of functions as well.

[00:33:52] JM: Why not just make like a functions API and just paper-over AWS Lambda or Google Cloud Functions or whatever?

[00:34:01] PC: Yeah. For the very simple reason that it's not open source. So if we wanted to go down that path, then we would actually end up using some sort of open source equivalent where you can install it yourself and we can support the community. So it would be an easy solution of course. We'd love to take the easy path that we just add a hook on top of it. But we really want to make sure that each part of our service is open-source and we'd support open-source communities as much as possible.

[00:34:28] JM: No. No. But I was just saying you could have an API that just sits over Google Cloud Functions or AWS Lambda or whatever. Like I think there are open source function as a service APIs that just paper-over those things.

[00:34:46] PC: The more interesting thing is the function as a server backend open source. So we really want to deliver on our promise that everything will be supporting open-source communities, and it's a bit cheating if we just go with, yet again, another cloud provider that you couldn't migrate away and you have to use AWS or some other cloud provider. We'd love to get to a solution where you choose your cloud. It doesn't matter. We'll just run it anywhere. But also, you can just run out on your own server. So if you want to install it on to your local machine and just run it there, then that's also a possibility. So that's where we need to get to. And definitely the functions is one of the hard ones along with storage of solving this. That's why we're doing a POC on the database functions to start off with. But I imagine in the long run for scalability, we will have to come up with a solution for the middleware and probably as a functions as a service type provider.

[00:35:48] JM: One of the useful aspects of Firebase is how easy it is to explore and just the rich GUI functionality. Are you trying to replicate that as well?

[00:36:00] PC: Yeah. In fact, I would hope that people can check this out. But I think our GUI experience might even be already a bit better in terms of data exploration. So they've got of course a JSON store, which have kind of got these mili-columns for exploring the JSON data.

Ours, because it's relational databases, it functions a lot more like a spreadsheet or with tabs. And if you've ever used Airtable, very much like that.

So we have this table view. It's a hidden away, because we've just been developing. But it's actually very stable now. We're going to make it the front and center of our dashboard very soon. So you can actually dig into your relationships. You can edit your data directly. You can create tables. You can create columns. You can add rows. You can even edit JSON data, because each column can be a JSON-B column. So we even provide mili-columns just like Firebase for JSON-B data or JSON data. So yes, we definitely have this functionality.

[00:37:00] JM: Is there anything else you'd like to add about Supabase? What else would be interesting to explore?

[00:37:05] PC: I think what would be quite interesting maybe is the types of people coming into the platform. What we're seeing maybe in the long run where we're going. So right now we've got basically two types of persona coming in. We either have people from the Jamstack platform. Actually a lot of the early developers came from the Dev 2 community, which I believe the founder very well. We actually have – I probably have to give him a shout out. I think of this 50 alpha customers came from an Tweet of his. We would release each month our updates for the month. Sort of our release notes on the Dev community. And then he sort of tweeted about one of ours, and we had a lot of sign-ups from him. The first 50 sign-ups, I think right when we're starting. So we've very basic of it at that stage.

This is one of the personas basically coming in. We have another type, which is the very engineering-heavy, people who like the Postgres environment and they want to get started. But over the long run, we're starting to see a lot of sort of agencies and enterprise customers coming in looking for us. So if you are an enterprise customer, what I would say is that basically once we get into beta, it starts becoming a lot more interesting. We've got all our security lock down. We've got performance and stability engineered into the platform.

If you are using Firebase and you're struggling, then I think then it becomes a very interesting proposition if you want to jump in and start using Supabase, then yeah, we'd be more than happy to work closely with these type of people to make sure that you've got a great

experience. And then we start engineering the platform specifically for your needs. Because we're so early on, we'd love to chat to these sort of people to figure out what exactly they're looking for and how we can build it into the system.

[00:38:54] JM: Well, thank you very much, Paul. It's been great talking to you, and interesting to know by your community as well.

[00:38:58] PC: Yeah. Yeah. I love for people to jump in. Actually, all of our community is centered around our GitHub. So if you jump in, you can see we've got GitHub. If you want to jump in and start commenting. We just got it actually this week. So very excited about that. As well, you can find us on Twitter, Supabase_IO. And if you start using us and you really like Supabase and you think you're going to build with it, feel free to reach out to me and I'll add you directly into our private company Slack and we'll work with you to make sure that you've got a great experience.

[END]