EPISODE 1232

[INTRODUCTION]

[0:00:00.0] JM: Have grown exponentially in importance since most of the world went remote. Basically, every popular online forum, message board, chat app and other online social aggregator was created before this new normal.

Many of these platforms lacks sufficient organization, or are just outdated for a fully remote environment. If society continues to work fully remote, or hybrid remote, then digital communities need to be optimally designed for remote experience. Tom Kleinpeter and Viraj Mody are two founders of Common Room, a platform made to bring communities together and develop in today's remote environment.

More importantly, Common Room is useful for aggregating all the different communities across a company's infrastructure; GitHub, Slack, Twitter and more. Common Room is in its early stages of development and its details have remained fairly secretive. In today's episode, Tom and Viraj unveil a lot of the details of the company. They discuss challenges with the current social platforms and what we can expect in the first version of Common Room, as well as some of the engineering and design decisions. Previously, Viraj led engineering organizations at Convoy and Dropbox, and Tom was a principal engineer at Dropbox.

[INTERVIEW]

[00:01:12] JM: Guys, welcome to the show.

[00:01:14] TK: Thanks. It's great to be here.

[00:01:15] VM: Thanks.

[00:01:16] JM: You both work on Common Room. I'd like to first explain what Common Room is, before we dive into it in the technical details that I'd like to. My rough overview would be for any given company, if that company has a community associated with it, like if it's a developer tool that sells into a community, that community has a Slack channel, a Discord channel, a

Twitter, any number of these social platforms. Common Room aggregates those. That would be my high-level description. Maybe just to drive the point home since people probably haven't seen it, maybe you guys can explain what Common Room is in your own words.

[00:01:57] TK: Yeah. We see online communities as being just a really, really important part of organizations. As you mentioned, this can take many forms. It can be spread out across lots of different pieces of software and sites online. Like you say, whether it's Twitter, or Discord, or Slack, or even something like a mailing list.

If you're building a product, having a well-run community can really help you achieve a lot of your goals. The people in the community can get answers to questions that maybe the company can't respond to you very quickly. People can also see that just get social proof, that this is a real product that people are using and getting value out of the organization, the company can get rapid feedback about what people like, what people don't like. You get bug reports very quickly through community sometimes.

Really, it lets the company build relationships that might not otherwise exist. You can identify people in the community that might be potential evangelists. You can make people feel welcome, feel they're a part of something. Common Room is software that's built to help organizations connect with their community and really deepen those relationships. Like I mentioned, finding evangelists is one example that also lets you keep track of people, like who have we sent t-shirts to? Maybe, who we should send a t-shirt to, or something like that. That's the problem we're going after. We just got started in really, August of last year. We've had a pretty exciting last six months, really taking it from zero to what we've got right now.

[00:03:25] VM: It's not enough to have conversations with the community. Everybody understands community voice and customer voice. People talk about it, but there isn't really anything out there that's purpose built to help companies allow their customers, or the community to feel like an extension of the team, or a part of the brand. That's what we're trying to build with Common Room.

[00:03:46] JM: Let's say, I am a gigantic company, like I have no idea if Twilio uses you. Twilio has a gigantic community of developers. Twilio has got a Slack channel, maybe a Discord

channel, GitHub, they've got all these different things. How would a company as largest Twilio, like what would be the onboarding process to use Common Room?

[00:04:10] TK: Yeah. The onboarding is fairly simple. Most of the providers we work with have pretty straightforward API integrations, where once you sign up with Common Room, you can sign in, you can run through the OAuth flow and get Slack and get Discord and GitHub all set up pretty quickly. Then at that point, you can start getting insights and visibility into your community, see what people are doing across the different providers. If someone has complained about something on Slack several times, then they went and created a PR on GitHub, you can correlate those activities, which is pretty interesting. The onboarding is pretty straightforward. You just sign up, click through some OAuth flows and you're in.

[00:04:51] JM: Once I do that, and Common Room is going to generate the aggregation of all my social accounts. What's happening on Common Room's side to do that?

[00:05:01] TK: Common Room is going to start getting a lot of webhook calls. Most of the APIs we integrate with are webhook-based for Discord, or Slack, or something like that. Once you sign up and install the app, we'll start getting webhooks with events that will start flowing into our system and then we build derived data on top of those events that help you tie it all together.

[00:05:24] VM: The one thing that has happened when you do hook it up though, there's an initial ingestion period of historical information. There's importing what is already on these disparate community platforms into Common Room, while also listening for future events via webhooks, or other similar API.

[00:05:44] JM: Are there any engineering challenges in that initial sweep of gathering data from all these different social accounts?

[00:05:51] TK: I think, the engineering challenge is more about the handling events going forward, importing the old stuff. It's a script. You run it. It calls the APIs. There's nothing too interesting happening there.

[00:06:06] VM: I mean, the only interesting thing we want to take care about when we're doing the ingestion is making sure we honor rate limits and batching and importing, because most of

these providers have constraints that we have to build into how we import. Yeah, I agree with Tom. Most of the challenges are on the other side.

[00:06:25] TK: I guess, one challenge is it's always a little weird saying this, having worked at companies with orders of magnitude, more data involved. Some communities have millions of events and then it's beyond the scale where you – I guess, you have to think a little bit about your indexes and things like that just to make sure we can still query everything very quickly. It's definitely not a Google or Dropbox scale data problem.

[00:06:50] JM: There have been other products that have done this social aggregation, I think. Are there any notable differences, or are there some critiques you had of previous social aggregators, or differentiation you have with Common Room?

[00:07:06] VM: From my perspective, the biggest difference is what we intend to do with the data and how we actually convert these data into actionable insights for the company, or allow them to derive value from the data. I can't think of too many people who've been able to connect dots the way we aspire to both in terms of who the community are, but also what the content being generated across all of these different platforms means to the company and how they can leverage it to build better products, or find the right problems to solve.

[00:07:42] JM: Give me a overview of the stack of Common Room. What are the programming tools that you use to create it?

[00:07:47] TK: Yeah. It's actually been pretty exciting starting with nothing last year and getting to pick all these choices. It can be difficult sometimes to make the right calls, but it's also fun to go and find what the best stuff is right now. I would say, overall, we bias towards boring stuff. We're not doing anything too crazy, nothing too new and exotic. We're trying to minimize variance and make technology choices that let us get the job done without having to really break any new ground, or potentially run into bugs that other people might not have had.

One of my tests for new technology is, can I go onto the Internet and find answers from people that have had similar problems? The place that started for us was what language are we going to go with and we ended on TypeScript there. It was important to me that we had just one

language. For our stage of the company, we have about 10 engineers right now. It's nice for people to be able to move around. There's this term, developer fungibility that floats around in my head.

Some of the best engineers that I've worked with have been people that can really work across the codebase. You might have a front-end engineer that can go dig into why a query might be slow. You can have a back-end engineer that can put something onto the admin page to make it easier for other people to diagnose things. Having one language just makes that easier and it also increases the leverage for any investments you make related to your development environment. If somebody adds a lint rule to prevent some error they've run into, everyone at the company can benefit from that.

We've thought about everything and landed on TypeScript. I'm also a big fan of typed languages. When I started at Dropbox, everything was on typed to Python. I saw the migration there as we went through and typed millions of lines of code. It just makes life dramatically better when you can turn runtime errors into your red squiggles in VS code. Code is just read so much more than it's written as an organization gets bigger. Yeah, TypeScript, I knew we could do everything with it. I guess, TypeScript and node really check the boxes of type language, where we can move problems to the left. Code problems, and code solutions in the type system and just find errors as quickly as possible.

Also, the languages were popular enough that it's easy to hire people and it's not that hard to get a language expert for TypeScript compared to something more exotic. I always feel it's really, really valuable to have someone on your team that is a deep expert in whatever language you're using.

TypeScript and node is where most of the magic happens. The front-end is React. It's modern React, but we're using hooks for everything. It's really nice to have just a clean code base there with the modern approach. We're using TypegraphQL for all of our API queries. That was one of the baits we had was whether we go with GraphQL, or Rest. Ultimately, it seemed like GraphQL was going to let us iterate a lot more quickly when we're building out front-end experiences. That that seemed like a win. As much as possible, we pay attention to developer velocity when

we're making these decisions. GraphQL just seemed like it would be a little bit more flexible there.

For storage, we're using RDS, PostgresSQL. I haven't had a ton of experience with no SQL stuff, but I've spent a lot of time working with relational databases. I like having my schema provide some level of enforcement, foreign keys and things like that just to help keep everything safe. Then the whole thing runs on AWS. We're using Pulumi for our infrastructures code, which has just been so nice. It's so good to be able to just set up a whole new stack, or make a change, just run a script and it goes and make, that does everything for us. Pulumi lets us describe all the infrastructure in TypeScript, rather than having to learn terraform or something like that. Engineers can look at the Pulumi files and see, it's a little easier to read and to see what's going on? That's the big chunks of the stack for us. Did I miss anything?

[00:11:55] VM: No, I think that was it. One of the biggest investments we've made in making the type system work through the back-end, and TypeGraphQL and into the front-end. That's been really nice to have a consistent type system and the data model represented across the entire stack without losing that context over the API boundary, which we invested some time in, but I think it was worth that investment.

[00:12:19] TK: Yeah, definitely.

[00:12:20] JM: For those who don't know, we've done a show on Pulumi. Maybe you could dive a little bit deeper into what problem Pulumi solves and why it's valuable to you.

[00:12:28] TK: Yeah, absolutely. AWS gets really complicated. There are 6,000 9,200, some very large number of things you can set up there. While one of my engineering goals, or principles is use as few primitives as you can, we still have database, Redis, some cues, some clusters of machines, we run our containers in Fargate, a couple EC2 machines, things like that. It's a non-trivial amount of stuff.

What Pulumi lets you do is describe everything you want with code. There's multiple solutions to this. It's in the general category of IAC, infrastructure as code. Pulumi is nice, because you actually write a TypeScript file. The way you make a TypeScript file that turns into a cue, or a

database, or a security gateway on the server, or in AWS is you just type something like, server equals new AWS.PostgresSQL server, or something like that.

You create objects in your TypeScript code that indicate the things you want to exist in the cloud. Once you have this described, it makes it very easy to make changes to it and spin up multiple copies of it. For instance, once we have our development infrastructure described, when we launched our production environment, it was just a few clicks. A minute later, we had a copy of our development environment that was called prod. I knew exactly what it was like.

Now going forward, when I want to make a change, change some security gateway, add a new cue, add a new database, I can just change the code, run it on the development environment, and then validate everything looks correct. Then when I run it on a production environment, I know exactly what's going to happen. I have just a good understanding of what I'm going to get. This also lets you take infrastructure changes and put them through code review. If I was configuring all this just by clicking around in the AWS console, I'd either have to have somebody on VC watching me, or it just wouldn't get checked.

This way, we can have reproducible behavior that also people can look at. It just makes everything so much nicer. I highly, highly, highly recommend if you're starting a new AWS project, use something like this to describe it. You will not regret it.

[00:14:55] JM: All right. Just to come back to the product and what you're trying to build. Who is Common Room for? Who on in average company is using Common Room?

[00:15:07] VM: I think we have a couple of personas that would benefit from the value we provide at Common Room. Most modern companies have community teams. They're either on the marketing side of the fence, or on the support side of the fence. Generally, they span both of those organizations in terms of how they operate and what they do. Part of it is if you have people in your community who are running into issues, how do you surface them and how do you get them help? Part of it is how do you find the champions? Who are in your community and engage with them, and use them as speakers for your conference, or have them be early adopters of some beta software. Those are the two primary buckets.

There's a third persona here that is on the product and engineering side of things, or like the product development side of things, where it's a really convenient way to find people who have feedback about specific features, or who may be really good user focus group participants for something, a PM, or an engineer has been thinking about adding to the product. It's a really good way to identify people in your community, who might have tangible and valuable feedback that helps you improve your products, instead of the conventional method of sending out a survey to a bunch of unknown users by the thousands and then waiting to see who responds and using those. Across all of these organizations in the company, there are different ways to optimize how you engage with your community, but each of them has its own unique value.

[00:16:41] JM: Do you expect it to fit into product cycles, like support, or sales, or marketing, or you think it's just going to be this broadly useful tool?

[00:16:55] TK: One thing we see is that a lot of people in the community function at companies, function as routers to some extent, where they are taking in what's in the community, which can be a very broad set of things and sending it to the people that need to know about it. Sometimes that's sales, sometimes that's engineering, because there's a bug. Sometimes it's product, because maybe they've seen some persistent issue that people have a hard time understanding something, or things like that.

We think there's a pretty broad use case. One of the things I want to accomplish with the Common Room is really helping to package up the insights that you can get out of a community, where with a vibrant community, you have a lot of information coming in about how people are using the product and how people are engaging with it and how people are misunderstanding it.

You can always tell, a good PM, or a good product thinker will look at the questions people ask and understand what's going through their head to cause them to ask that question. What are they not understanding? What are they thinking about? There's also a lot of noise in communities. There are people that are complaining about things that are already fixed, or just talking, or different stuff.

One of the things I would like for us to be able to do, and we're not there quite yet, is really have very information dense distillations of things that are happening in the community that might be

useful for somebody who is trying to set the roadmap for a product; things that they might care about.

[00:18:30] VM: The things I mentioned about the community team being the router, is really true. What is tough today is they use all sorts of manual, janky processes to efficiently route things to different teams within the company. A lot of valuable information is lost, just because they cannot keep up. One focus with Common Room is how do we build the right workflows to enable these interactions to be unmissable? Also, just more efficient than trying to do all of this manually in a spreadsheet somewhere, using URLs to 10 different types of services.

[00:19:08] JM: What's been the hardest engineering problem you've had to solve so far?

[00:19:12] VM: It's probably going to be on the event ingestion log. I guess, while Tom thinks about it, I'll step one layer above and generally say, that the hardest problem we have is that data we get is not clean. Data from GitHub is different from your community in Slack and your community in Discord. From my perspective, creating a normalized base of consistent data that we can then use to convert into valuable insight would be my answer for the hard problem to solve.

I think this is one where it's really easy to blame the data in some sense, where it's very easy to find faults with how the data isn't right. I think our focus is really on making sure that okay, look, the data is what it is. How can we apply clever techniques to make it consistent, make it correlateable? That's also competitive advantage, in some sense. All energy invested there is valuable, because it helps us build a better experience for our customers.

[00:20:15] TK: Building an architecture that lets us ingest lots of content from lots of different providers and make sense of it, deal with things coming in out of order, being able to deal with mistakes we make, because we had a schema misconfigured, or something like that. Just getting this system set up to where we can rapidly add new providers. Because every time we finish one provider, support for one provider, customers ask for two more. It's endless.

Just architecting the system such that we can quickly add new providers, but also deal with mistakes we make, adjusting providers, replay data, things like that, I think has been the place we certainly spent a lot of time trying to make it work.

[00:20:59] JM: Are there really that many online community types?

[00:21:02] TK: I keep getting asked about ones.

[00:21:05] VM: What's interesting is your community exists in the real world as well. Companies will do events, and they'll do meetups. As companies start to understand the value they can derive from the online communities, the next question is, okay, well, how do I integrate my offline communities into this? There's platforms out there that support, empower companies to build communities in the real world. There's ways to integrate with that data. We're trying to figure that out, stuff like Bevy is out there, which is a great example of a in real-world community building, too.

[00:21:43] JM: Is the model for adding additional customers, each new user, just additional data is added to your database. Do you scrape the data and just store it? Are you periodically scraping the data from all these social sites? I'm just curious about how you keep the data fresh.

[00:22:05] TK: All the events that people care about are getting pushed to us right now, with the exception of CSV uploads. We support just uploading events via CSV is a catch-all for things we haven't implemented yet. For all the events that come in from our providers, those are all pushed directly to us. For things like tweets, we do have to query Twitter and pick up the fresh ones. We just do that on an interval.

[00:22:31] JM: Each new customer, you just store in a database. You just store it like – You're not spinning up a new instance of the app for new users, are you?

[00:22:42] TK: All the customer's data is kept logically separate. Every row is tagged with the community ID that they're associated with. It's stored in the same actual database.

[00:22:53] JM: Have there been any community providers that have been particularly difficult to engineer around?

[00:22:58] TK: GitHub is pretty complicated. There's a lot that can happen in GitHub, between PRs and issues and comments, replies, all that. There's probably the most code for dealing with GitHub right now. They have a really nice GraphQL API, which has been really easy to work with.

[00:23:19] JM: How do you see communities changing in the near future? I mean, I certainly see developer relations becoming more and more integral to how companies work. We just did a show on orbital and super orbital, or sub-orbital. I think it was sub-orbital. I can't remember which one does community developers, community relations. Clearly, developer relations and community are becoming more important. Do you have any perspective on how that's going to change in the near future?

[00:23:47] VM: I think for developer-focused companies, but also generally for most B2B companies, I think, doing community correctly, essentially makes your community a valuable extension of your company. At some point, there's only so quickly you can grow and doing community correctly allows you to leverage the power of your best customers and your champions out there. You have all these people in our community who graduate from being passive buyers of your product, or having passive engagement to then having active engagement as a first step. Then, actually becoming evangelists for your product, or your technology. I think if we do it right, that progression becomes faster and the ability for companies to scale their presence increases exponentially via their communities.

[00:24:40] JM: Yeah. A lot of this seems driven by open source companies, but I think there's just as much significance around companies that don't have an open source presence, like Twilio, I know has a huge community, even though it's not open source software. Are there communities that are too big? If you look at AWS, AWS has a community, but it seems like AWS would be too big to use Common Room. They probably wouldn't want to.

[00:25:08] VM: There are companies beyond the developer realm. Twilio is a good example of still a developer-focused company. There are other productivity companies, like Figma, for

example, that focus on another whole segment, but have really strong community-led motions. AWS is an interesting example, because you can almost think of AWS as a collection of smaller communities. Each of those could definitely benefit from the motions that Common Room could make successful.

Yeah, we don't think of AWS as too large, mostly because we have both the capability, but also, AWS itself is logically partitioned, so that you can slice and dice it how you want. In many ways, the magic here could really be seeing how your entire community is then mapping the overlapping circles that represent your entire community, but then are still partitioned across the various sub-sections. If AWS serverless is a separate entity from say, AWS storage, you can see who in your community overlap, how they overlap, if one is more engaged than the other. I feel all of those are in the realm of what we hope to do with Common Room.

[00:26:24] TK: Common Room was actually founded by – our CEO is Linda Lian, who worked at AWS and was in charge of building community for I think, the serverless.

[00:26:35] VM: Serverless. Yep.

[00:26:37] TK: She was frustrated that there was not a solution available on the market for her, and so she left to start Common Room. I would definitely aspire to help AWS with their community at some point. Yes.

[00:26:49] JM: When the user OAuths into all of the community services in Common Room, or the different community providers, is there a security concern there? Are there difficult security problems you have to work on?

[00:27:03] TK: In general, we get the lowest permissions possible. The apps you set up with these, with the more modern versions of them, they have a very flexible scoped API model. We go with the lowest level of permissions we can get away with. There's also security reviews, lots of companies that we go through when we're onboarding them. There's definitely security concerns, so we try to be very careful and have minimal data. If I lay awake at night thinking about anything, it's about security. It's how we're keeping our servers safe.

One of my big rules there is just the simpler you keep everything, the easier it's going to be to keep it secure. The less AWS stuff we have, the better, the less surfaces, the less permissions. Security is super important. I think we're doing a pretty good job with it so far.

[00:27:56] JM: When you integrate with a Slack, do you scrape the messages from Slack? Or do you have some aggregation of engagement?

[00:28:03] TK: We get the message content, so that when you're looking at our site across what is happening, you can see, just in one feed. It's easy to scroll through, in a way that is very fast. We don't have to fetch those on demand from Slack.

[00:28:20] JM: Common Room has been in stealth for a pretty long time. When this is coming out, it's going to be your unveiling of a product that has been in stealth. Do you have any advice around how to build a product while remaining in stealth?
[00:28:35] VM: The reason we're in stealth is generally, it's nicer to have something to show and actually talk about people who are getting value from it, versus talking about an idea of out of it, or write vaporware. My advice would be being in stealth is not the goal. Some other people may think of being in stealth as something valuable for another purpose. For us, it was mostly about making sure that when we talk about what we're doing, we can actually not only demonstrate the value it provides, but also clearly illustrate how our customers are using it to solve real problems.

Just because you're in stealth, doesn't mean you can get actual customers and you cannot solve real problems out there. We don't really shy from talking about Common Room when we have to. It's mostly, I'm too conservative to go sell vaporware before I can go show something, is how I think about it.

[00:29:32] TK: The first lines of code for this service were written in August of last year, so rough six or seven months ago, seven months. Geez, it's already March. We had customers using the product by November, I think.

[00:29:46] VM: Yeah. I think, mid to late November.

[00:29:48] TK: We definitely were not one of these stealth companies that goes and works on the secret masterpiece for four years, without any customers, because that is a complete anti-pattern. I think, the most important thing about being in stealth is it's not an excuse to not be super engaged with your customers.

Ultimately, your job is not to build software. It's to build customer value. The only way to really know that you're doing that is to actually put something in front of customers and see how they think. We've definitely made significant changes to the product since November, when we started showing it to people that we had something we thought was cool internally, but then we show it to people and they want changes about it, and we made the product better. Being in stealth is a – it's a way of simplifying your world. You don't have to worry about as much. It's not an excuse to avoid dealing with customers, because customers are the only thing that really matter.

[00:30:43] JM: Your team is really strong. You've got a lot of a lot of experience across the board, a lot of ex-Dropbox, like you said, ex-Amazon people. I'd love to know when you have a really strong founding team, like you do, how have you structured the company? How do you run meetings? Do you do anything organizationally that might be of interest to the listeners?

[00:31:06] VM: I think our North Star is really maintaining velocity and not losing our customer focus. Generally, how we organize, how we operate, how we execute revolves around those two principles. Everything Tom said about being in stealth, not being an excuse to not engage with customers applies even to how we think about internally in terms of ownership of features. It's always about ownership of problems, right? We have a pretty strong engineering team. It's never about go build this feature. It's about here's a problem our customer has, understand it and solve it. Then, all the classic things around lessons we've learned from the past of how to build a high-performing team that has strong velocity, and then making explicit choices in order to enable that.

[00:31:57] TK: One thing that's really important to me at this stage of the company is building a very strong bench of engineers that can just handle very open-ended problems. We're 10 engineers right now. We're going to be a lot more than 10 engineers in a year or two. The

founding people we have now, the set of engineers we have now, I want them to grow a great deal over the next couple of years.

I spent a lot of the first 10 years of my career on tiny teams, very small startups. I spent a lot of time taking things that didn't exist and making them exist. I think it's a very, very useful skill that's rare in bigger companies. At a larger company, engineers are frequently working on something that already exists. There's a lot of guardrails around what they're doing. Startups are really the perfect training ground for how to wrestle with ambiguity, how to take something that doesn't exist, that a customer wants and make it happen.

As I think about the way we build things, it's important to me that the people at the company now and the people that we hire coming up, get the chance to take open-ended things and go make them happen. I'm trying not to over specify how we build things here. I like to be able to tell people we need search. Let's go make it happen. Go figure out what you can figure out. It's okay if you make mistakes. I got your back. We'll fix them quickly. When we get done, we'll have search and we'll also – you'll have experience solving the hard problems that gets harder to get at a bigger company.

If you do that, if you have people that can independently solve problems, you can really move quickly. Because one of the big things that slows you down is just when you need to build a lot of consensus around something, when what could be – instead of just opening up the VS code and sending out the diff, you call a meeting, or you schedule a meeting for two days from now to talk about how we're going to do something. I want to build a corps of engineers that can be decisive, have confidence in their decisions and go make things happen.

[00:34:08] VM: From a organizational perspective, what our job becomes is empowering them, so that they have the ability to make these decisions and execute. If decisions need to be made, our job is to be decisive. The worst thing we can do is some engineer comes to us with a question about something and we send them on a wild goose chase to get more data, or try something new.

A lot of leading by example, in terms of distilling a plan from ambiguity and then thinking about how to make trade-offs. Velocity, I think, is also a direct output of high-quality trade-off decisions.

Because generally, everything we do in any dimension in technology is about choosing what side of the pros and cons you want to be on. Part of how we operate as a founding team is how help our team understand how we think about making those trade-offs, instead of making a trade-off for them.

[00:35:05] JM: All right, I guess just to close off, how do you guys see the next two to five years of Common Room looking like?

[00:35:11] VM: From an organizational perspective, I think the surface area where we can have an impact is starting to – it's pretty clear that it's huge. The classic hypergrowth thing is something we're going to have to deal with. I feel fortunate, having dealt with it a couple of times in the past and having strong partners, and Tom and Francis and Linda, where we've all experienced this in different ways. Hopefully, we'll take some of the pain out of hypergrowth. That's the thing I suspect is going to keep me busy for the next few years at least.

[00:35:46] TK: Yeah. I think we're going to grow very quickly. I think it's going to be really fascinating to see all the choices we made in the last six months, how they age, how they break down. It's going to be certainly an educational experience. It'll be fun to check my notes in a couple years and see how everything went.

Yeah, we're just excited. I've worked for companies that didn't hit product market fit, that we did great engineering and we built something that we thought was great, but some people cared about it, but just not enough. This feels much more like rolling a rock downhill. We have found something that people want. Now, we get to do the fun part of refining it. The work is not remotely done. There's still a ton to do. I think we're going to see a lot of graphs going up into the right, which is just really energizing for everybody on the team. I think we're going to have a lot of fun scaling problems and just product problems as we go forward.

[00:36:45] JM: Okay, guys. Well, thank you for coming on the show. It's been a real pleasure talking to you.

[00:36:48] VM: Thank you.

[00:36:48] TK: Yeah. Thank you so much.

[END]